IOPE: Interactive Ontology Population and Enrichment guided by ontological constraint (Technical Report)

Shadi Baghernezhad-Tabasi, Marie-Christine Rousset, Loïc Druette, Fabrice Jouanot, Celine Meurger

IOPE: Interactive Ontology Population and Enrichment guided by ontological constraint (Technical Report)

Shadi Baghernezhad-Tabasi[1], Marie-Christine Rousset[1,3], Loïc Druette[2], Fabrice Jouanot[1], and Celine Meurger[2]

[1] Université Grenoble Alpes, CNRS, LIG, Grenoble, France
firstname.lastname@univ-grenoble-alpes.fr
[2] Université Claude Bernard Lyon 1, SAMSEI, Lyon, France
firstname.lastname@@univ-lyon1.fr
[3] Institut Universitaire de France, Paris, France

Abstract. In this paper, we focus on the construction of specialized ontologies that capture skills of experienced experts in a particular domain with the goal to share them with a larger community of trainee or less experienced experts in the domain. Our main contribution is the automatic construction of a Graphical User Interface (GUI), built from the ontological constraints of a given ontology, as the support of the controlled update process of the considered ontology. In an extensive set of experiments we discussed the effectiveness and efficiency of our proposed approach in a specialized medical domain.

Keywords: Ontology update · Graphical User Interface

1 Introduction

Ontologies on the Web are the backbone of many information systems that require access to structured knowledge. Recently, with the arrival of Linked Open Data sources like DBpedia [5], and by Google's announcement of the Google Knowledge Graph in 2012, graph-based representations of general world knowledge have drawn a lot of attention again. There are various ways of building such knowledge graphs. They can be curated like Cyc [13], edited by the crowd like Freebase [20] and Wikidata [24], or extracted from large-scale, semi-structured open data or web knowledge bases such as Wikipedia, DBpedia [5] and YAGO [19]. By their very nature, real world ontologies are dynamic artifacts that evolve both in their structure (the data model) and their content (instances). *Ontology enrichment* is the task of extending an existing data model of an ontology with additional concepts and semantic relations, while *ontology population* is the task of adding new instances of concepts to the ontology.

In this paper, we focus on the construction of specialized ontologies that capture skills of experienced experts in a particular domain with the goal to share them with a larger community of trainee or less experienced experts in the

domain. This is the case in particular for domains related to pedagogy because teaching objectives are hard to formalize and teaching methods are difficult to share within a common and standardized referential. For instance, OntoSAMSEI [6] captures how to design teaching units for learning skills by simulation in Medicine. OntoSAMSEI has been bootstrapped based on the reported expertise of a group of pioneer trainers who have documented simulation learning units of various types that they have introduced in their teaching. As a result, 30 different types of simulation sessions have been formalized in OntoSAMSEI by making explicit for each of them the target audience, the aimed objectives, the prerequisites, the resources required (human, consumable, simulator, material), as well as the evaluation mode of prerequisites and objectives. The resulting ontology is a hierarchy of classes and of properties enriched by *ontological constraints* on the properties and on the classes that convey the constraints that will have to be fulfilled by their future subclasses, subproperties or instances.

In this paper, we show how to exploit such ontological constraints as a source of guidance for (possibly less experienced) teachers willing to design their own simulation sessions. Our main contribution is the automatic construction of a Graphical User Interface (GUI), built from a given ontology, as the support of the controlled update process of the considered ontology.

The paper is organized as follows. Section 2 describes the formal background of the ontologies that we consider. Section 3 describes our methodology for the automatic construction of a GUI guided by an input ontology, and its usage for guiding its update (population and enrichment). Section 4 summarizes the evaluation conducted to assess the added value of the GUI for ontology updating. Section 5 is dedicated to related works while Section 6 concludes the paper.

2 Formal background

An ontology is a shared formalization of a a domain of interest based on a structured vocabulary made of classes, properties and instances. Ontological constraints are first declared on classes and properties to constrain their formal semantics to fit with their actual meaning in the domain of application. Then, factual statements can be added to describe specific entities as instances of classes with specific values for some properties. The ontological constraints are defined in RDFS[4] and OWL[5]. Following the Semantic Web [4] and Linked Data [10] standards, both the ontological constraints and the factual statements can be uniformly described in the RDF format[6].

2.1 RDF format

Let I, L and B be countably infinite pairwise disjoint sets representing respectively *IRIs*, *literals* and *blank nodes*. IRIs (Internationalized Resource Identifiers)

[4] Resource Description Framework Schema (RDFS): `https://www.w3.org/TR/rdf-schema`

[5] Web Ontology Language (OWL): `https://www.w3.org/TR/owl-features`

[6] Resource Description Framework (RDF): `https://www.w3.org/TR/rdf11-concepts`

are standard identifiers used for denoting any Web resource involved in RDF statements. A literal is a string that represents a specific value for some properties. A blank node represents an anonymous resource (either a literal or an IRI) that can have a local identifier such as *_:b1*.

An ontology in RDF is a set of (factual or ontological) statements expressed as triples that form a graph whose nodes are IRIS, blank nodes or literals.

Definition 1. *RDF graph An* RDF graph *is a finite set of* RDF triples (s, p, o), *where* $(s, p, o) \in (I \cup B) \times I \times (I \cup L \cup B)$.

To simplify IRIs, RDF allows defining *namespaces* within an RDF graph. A namespace is an abbreviation of the prefix of a IRI. In particular, predefined relations are declared in RDF, RDFS or OWL namespaces like *rdf:type*, *rdfs:subClassOf*, *rdfs:domain*, *rdfs:range*,*owl:hasValue* or *owl:minCardinality* for specifying ontological constraints in a standard way.

2.2 Ontological constraints

The ontological constraints that we consider are RDFS constraints and some OWL constraints.

There are 4 types of RDFS constraints:

- **Class specialization constraints** denoted by triples of the form `(C rdfs:subClassOf D)` specify that a class `C` is a subclass of a class `D`, i.e., that every instance `i` of `C` is an instance of `D`: ∀ `i` (`(i rdf:type C)` ⇒ `(i rdf:type D)`)
- **Property specialization constraints** denoted by triples of the form `(p rdfs:subPropertyOf q)` specify that a property `p` is more specific than a property `q`, i.e.:
  ∀ `i` ∀ `j` (`(i p j )` ⇒ `(i q j)`)
- **Domain constraints for a property** denoted by triples of the form `(p rdfs:domain C)` specify that every subject of a property `p` is an instance of the class `C`, i.e.:
  ∀ `i` ∀ `j` (`(i p j )` ⇒ `(i rdf:type C)`)
- **Range constraints for a property** denoted by triples of the form `(p rdfs:range D)` specify that every object of a property `p` is an instance of the class `D`, i.e.:
  ∀ `i` ∀ `j` (`(i p j )` ⇒ `(j rdf:type D)`)

Figure 1 displays part of the specialization hierarchies of properties and classes resulting from RDFS ontological constraints declared in OntoSamsei.

We consider 3 additional types of constraints that are expressed as OWL restrictions on properties for classes:

- **Value restrictions**, that we will denote by `(p value v)` ∈ `Constraints(C)`, specify that every instance `i` instance of the class `C` must be related by the property `p` to the value `v`:

$$\forall \texttt{i}(\texttt{(i rdf:type C)} \Rightarrow \texttt{(i p v)})$$

Fig. 1. A part of hierarchy of properties and classes in OntoSamsei

– **List of values restriction**, that we will denote by `(p rdf:Bag [v1, v2, ..., vn])` $\in$ `Constraints(C)`, ...
– **Cardinality restrictions**, that we will denote by `(p min k D)` $\in$ `Constraints(C)`, specify that every instance `i` instance of the class `C` must be related by the property `p` to at least `k` distinct instances of the class `D`:

$$\forall \texttt{i}(\texttt{(i rdf:type C)} \Rightarrow \exists o_1, \ldots o_k(\bigwedge_{i,j \in [1..k]} o_i \neq o_j \wedge \bigwedge_{i \in [1..k]} (o_i \texttt{ rdf:type D}) \wedge \texttt{(i p } o_i\texttt{)}))$$

These two types of restrictions can be written in RDF using several RDF triples. The RDF graph corresponding of the shortcut notation `(C: p value v)` is made of the following RDF triples:

```
(C rdfs:subClassof _:b1) (_:b1 rdf:type owl:Restriction)
(_:b1 owl:onProperty p) (_:b1 owl:hasValue v)
```

Similarly, the RDF graph corresponding of the shortcut notation `(C: p min k D)` is made of the following RDF triples:

```
(C rdfs:subClassof _:b2) (_:b2 rdf:type owl:Restriction)
(_:b2 owl:onProperty p) (_:b2 owl:onClass D)
(_:b2 owl:minCardinality k)
```

2.3 Illustrative example

For illustration purposes, we visualize some of the ontological constraints defined in OntoSamsei ontology.

We show the RDF graphs corresponding to three OWL ontological constraints declared for the class `samsei:PortACathPlacement` which is a particular type of simulation learning unit that trains students to place a a port or a catheter. These constraints are declared for two properties describing the required resources for conducting this type of simulation-based training session, namely `samsei:equipmentSupplies` and `samsei:simulatorResources` which are specializations of the property `samsei:resources`.

Figure 2 shows the RDF graphs associated to two constraints declared in the ontology on the property `samsei:equipmentSupplies` for the class `samsei:PortACathPlacement`, which is a particular type of simulation learning unit that trains students to place a a port or a catheter.

The constraint graph in Figure 2(a) expresses that `samsei:sterilecompress` (which is an instance of Bandage material) is declared in the ontology as a mandatory value of the property `samsei:equipmentSupplies`.

The constraint graph depicted in Figure 2(b) expresses as an additional constraint that at least one equipment of type `samsei:protectiveSupplies` is mandatory for simulating a placement of a port or a catheter.

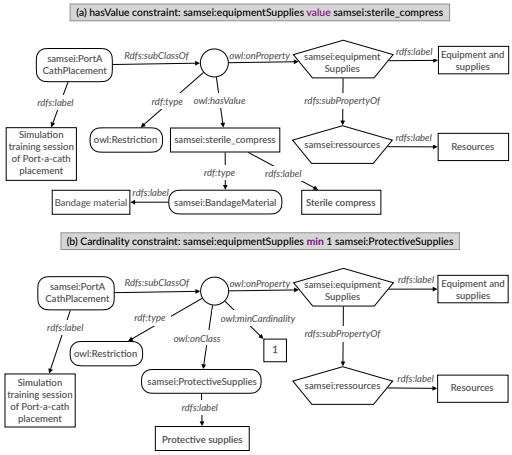

Fig. 2. Two constraint graphs (a) and (b) on the property `equipmentSupplies` for the class `PortACathPlacement`

Figure 3 shows the constraint graph associated to a cardinality constraint declared in the ontology for the property `samsei:simulatorResources`. The constraint expresses that at least one simulator of type `samsei:VenousChestSimulator-Manikin` is mandatory to train students to place a port or a catheter on the right spot of the patient body.

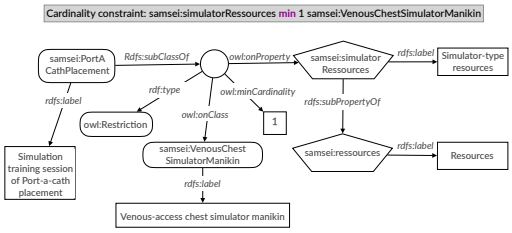

Fig. 3. A constraint graph on the property `simulatorResources` for the class `PortACathPlacement`

3 Interactive Ontology Update

In this section, we describe the interactive approach that we propose for enriching and populating ontologies by involving users in the process. The users that we target are domain experts that are not familiar with ontology formalization and engineering. In particular, they may not know the RDF format and the machinery underlying the different components of an ontology.

3.1 Methodology Overview

Our approach consists in transposing the RDF data and the ontological constraints of a given domain ontology into a graphical user interface (GUI) named IOPE GUI. It functions as a guidance for domain experts to easily explore the ontology and update it through interactive graphical widgets. The input entered by domain experts through the IOPE GUI are transformed into RDF triples that must be verified by an expert in ontology engineering before being added effectively in the domain ontology. Figure 4 is an overview of our interactive IOPE system.

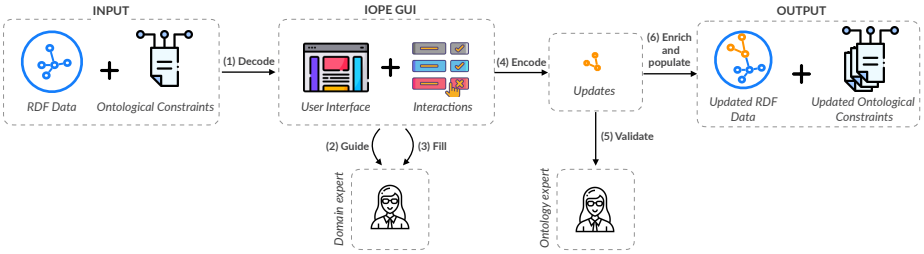

Fig. 4. Overview of IOPE workflow

The IOPE GUI is made of pre-filled Web pages that are automatically generated to reflect the domain ontological constraints. For doing so, we have followed a declarative approach based on a set of *mapping rules* from *constraint graphs*

to *Web form templates* described using a Web form ontology called IOPE_Web that we have developed by adapting RAUL [9]. The IOPE_Web ontology is described in Section 3.2, while the mapping rules are described in Section 3.3. The data binding mechanism needed to bind the user input in the pre-filled Web pages to RDF data is performed by a set of *binding rules* that are described in Section 3.5.

3.2 The IOPE_Web ontology

The IOPE_Web ontology is shown in Figure 5. It is organized around 4 main classes `IOPE:Page`, `IOPE:PageLayout`, `IOPE:Container` and `IOPE:Widget` related by properties for modeling that Web pages are structured in containers with widgets and are associated with page layouts. The interaction with users

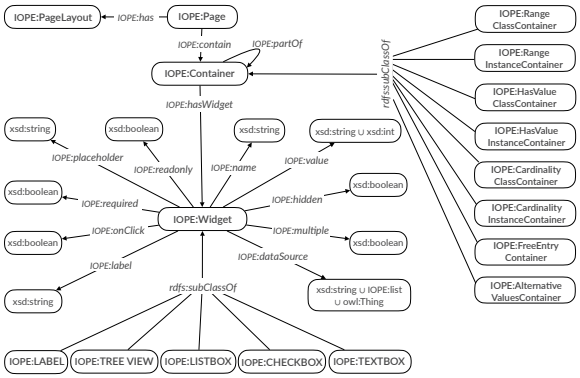

Fig. 5. The IOPE Web form ontology

can be done using several types of widgets such as LABEL , TREE-VIEW, LIST BOX, TEXT BOX and CHECK BOX, which leads to as many corresponding subclasses of the main class `IOPE:Widget`. They inherit of the standard properties for widgets that are described in IOPE_Web as datatype properties such as `IOPE:name`, `:placeholder`, `IOPE:dataSource`, and others. In our setting, the `IOPE:dataSource` property is used to assign an input data from a domain ontology that can be of type *xsd:string*, simple or nested list *IOPE:list*, or *owl:Thing*. The `IOPE:value` property is filled by the output value of the widget provided by a user. The boolean properties `IOPE:hidden`, `IOPE:multiple`, `IOPE:readonly` and `IOPE:onclick` are similar to HTML form attributes. A widget with the `IOPE:required` property as 'True' will be rendered by a red asterisk to specify that the widget must be filled by the user.

Widgets can be grouped in a Web page within containers that can be nested using the `IOPE:partOf` property. In our setting, different types of specific containers are declared as subclasses of `IOPE:Container` to express that the differ-

ent types of ontological constraints in our setting will be rendered in a specific manner in the IOPE GUI.

The ordering of the widget elements in the containers and in the pages are defined within a page layout. Figure 6 shows the page layout that is built at the initialization process of building the IOPE GUI for a given focus class F chosen by the user as her class of interest.

3.3 Ontology-based GUI Construction

Input: The input of GUI construction is a domain ontology in which the ontological constraints has been automatically saturated by a reasoning algorithm. Given the ontological constraints that we consider (see Section 2), the saturation of the constraints can be done iteratively as follows by a breadth-first traversal of the class hierarchy:

- For each class D, compute the set $Constraints(D)$ of all constraints holding for D by adding to the set of ontological constraints declared for D the ontological constraints declared for its super-classes.
- Simplify the resulting set of ontological constraints by deleting redundant ones. A constraint `( p min k c)` is redundant in $Constraint(D)$ if there exists in $Constraint(D)$ a constraint `(sp min k' sc)` such that ($sp = p$ or sp is a sub-property of p) and ($k' \geq k$) and ($sc = c$ or sc is a sub-class of c). For example, the (inherited) constraint *(samsei:equipmentSupplies min 1 samsei:Material)* is redundant with the (declared) constraint *(samsei:equipmentSupplies min 2 samsei:Syringe)* within a given set of constraints.

Initialization: The GUI construction is initiated with the choice of one class of interest in the ontology by the user, which is called the *focus class* F in the following.

The set $Constraints(F)$ of the ontological constraints associated to the focus class F is decomposed in groups $Group(P, F)$, where P is a property involved in atleast one constraint of $Constraints(F)$, defined as follows:

- If there is no constraint in $Constraints(F)$ involving sub-properties of P, then $Group(P, F)$ is simply the subset of all the constraints involving P in $Constraints(F)$, constraints involving the properties that are sub-properties of P.
- Else $Group(P, F)$ is the subset of all the constraints involving sub-properties of P.

In the remaining of the paper, for avoiding the description of too many particular cases, we focus on the latter more general case.

For the focus class F, and for each group of properties $Group(P, F)$, an instance of a Web page is created with the page layout depicted in Figure 6, which sets up the organization within the page of the specific containers dedicated to the different types of constraints on sub-properties p of P for which there exists constraints in $Group(P, F)$.

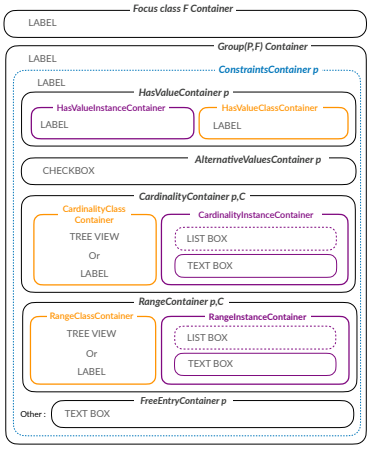

Fig. 6. Empty web page prepared for the rendering of constraints of the focus class F for each property p which is a specialization of a same property P.

The following instances of the *IOPE:Container* class are created with their position in the empty Web page shown in Figure 6:

- *IOPE:FocusClass F Container* denotes the main container of the created Web page
- *IOPE:Group*(P, F)*Container* denotes the container that will group all the containers corresponding to the constraints holding for F on sub-properties of P
- *IOPE:ConstraintContainer* p denotes the container that will display the restrictions of F on property p
- *IOPE:HasValueContainer* p denotes the container that will display the hasValue restrictions of F on property p
- *IOPE:AlternativeValuesContainer* p denotes the container that will display the list of values restrictions of F on property p
- *IOPE:CardinalityContainter* p, C denotes the container that will display the cardinality restrictions of F on property p and class C.
- *IOPE:FreeEntryContainer* p denotes the container for the user to add new classes involved in cardinality restrictions for the property p.

Then we employ a set of mapping rules which map components of each ontological constraints to the visual widgets in the prepared containers in order to fill each Web page guided by the ontology.

Mapping rules: Each mapping rule has a constraint graph pattern in its left-hand side and a IOPE_Web graph pattern in its right-hand side. The constraint graph pattern expresses a particular ontological constraint on the focus class. If it can be instantiated in the input data, the corresponding instantiation of IOPE_Web graph pattern in the right-hand side is a specification using the

vocabulary of the IOPE_Web ontology of how to fill the corresponding widgets and containers in the corresponding Web page.

The mapping rules can be triggered in a forward-chaining manner and in any order. The resulting IOPE_Web graph provides the full RDF specification of the pre-filled Web pages that have to be created for the focus class chosen by the user. The effective implementation of the mapping rules is implemented using RDFLib and JSON libraries in Python 2.7.16 language. Our code is publicly available in [1].

For clarity purpose, we will describe the mapping rules in their instantiated form where the left-hand side is a constraint graph in which F, p and P denote respectively a given focus class F, a particular property p sub-property of a given property P for which a certain number of ontological constraints are grouped in $Group(P, F)$.

The set of mapping rules are given in Annex. We just provide here one mapping rule of each type.

1. **Mapping rule for a group of constraints of a focus class F on sub-properties of a property P**. This mapping rule is presented in Figure 7. It applies for each group $Group(P, F)$ of properties, and each property p in

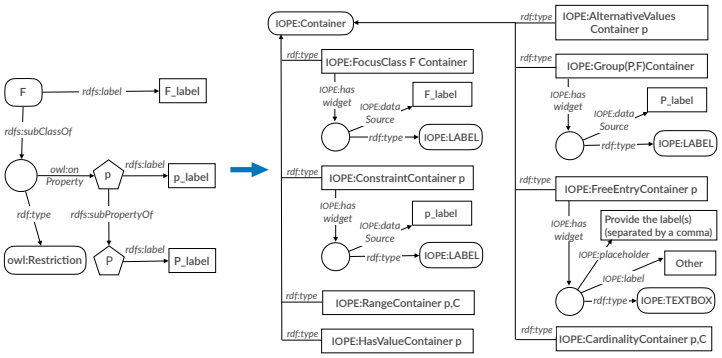

Fig. 7. Mapping rule for a group of constraints of a focus class F on sub-properties p of a property P

$Group(P, F)$ as follows:

- the specific containers denoted respectively
  - *IOPE:FocusClass F Container*
  - *IOPE:Group(P, F)Container*
  - *IOPE:ConstraintContainer p*

  are declared as instances of the *IOPE:Container* class, and widgets of of type *IOPE:LABEL* are created as blank nodes with the property *IOPE: dataSource* filled by the corresponding label of F, P and p in the domain ontology.

- The specific container denoted *IOPE:FreeEntryContainer p* is declared as an instance of the *IOPE:Container* class, and an associated widget of type *IOPE:TEXTBOX* is created as a blank node, to collect user future inputs concerning the property p.
- The three specific containers denoted respectively:
  - *IOPE:HasValueContainer p*
  - *IOPE:AlternativeValuesContainer p*
  - *IOPE:CardinalityContainter p, C*

  are just declared as instances of the *IOPE:Container* class. The widgets associated to them will be created by other mapping rules described in the following.

2. **Mapping rule for a value restriction** `(p value v)` **such that** `v rdf:type C` . This mapping rule is described in Figure 8. The specific container *IOPE:HasValueContainer p* is decomposed in two sub-containers defined as blank nodes whose types are *IOPE:HasValueInstanceContainer* and *IOPE:HasValueClassContainer* respectively. For these two sub-containers, widgets of type *IOPE:LABEL* are created as blank nodes with the property *IOPE:dataSource* filled by the corresponding labels of v and its class C in the domain ontology. The property *IOPE:required* is set to 'True' for the first widget to refer that the value v is mandatory for the property p.

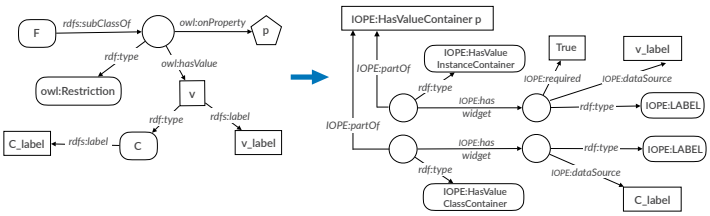

Fig. 8. Mapping rule for a value restriction `(p value v)` where `v rdf:type C`

3. **Mapping rule for a cardinality restriction** `(p min n C)` **such that** $n > 0$**,** `C` **has a hierarchy of sub-classes and a list of instances in the domain ontology**. This mapping rule is described in Figure 9.
   The specific container *IOPE:CardinalyContainer p, C* is decomposed in two sub-containers defined as blank nodes whose respective types are:
   - *IOPE:CardinalityClassContainer*
   - and *IOPE:CardinalityInstanceContainer*

   For the former, a widget of type *IOPE:TREEVIEW* is created as a blank node with the property *IOPE:dataSource* filled by the the tree view of $subClasses(C)$, which denotes the hierarchy of the sub-classes of C in the domain ontology enriched with an additional item $Other\_C$. The property *IOPE:required* and *IOPE:onClick* are set to 'True' for this widget to indicate that entering at least one value is mandatory for the property p and that

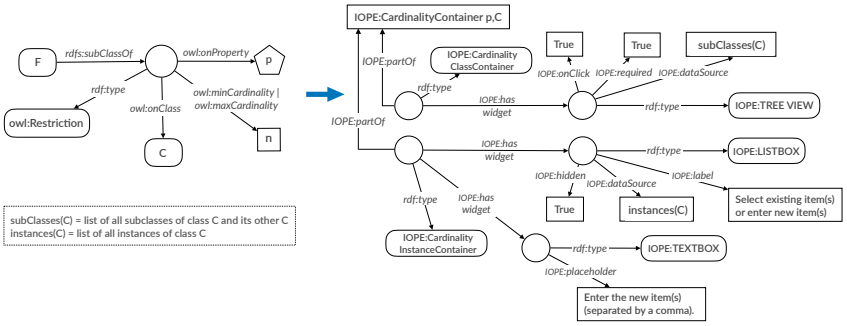

Fig. 9. Mapping rule for a Cardinality constraint where subClasses(C) and instance(C) are not empty

this widget supports the interaction with users to display interactively the sub-class hierarchy.
For the latter, a widget of type *IOPE:LISTBOX* is created as a blank node with the property *IOPE:dataSource* filled by the list *instances*(C) of instances of C , the *IOPE:label* property set to 'select existing item(s) or enter new item(s)' and the *IOPE:hidden* property set to True to make the widget invisible until the first interaction of the user through the widget of type *IOPE:TREEVIEW*. A widget of type *IOPE:TEXTBOX* is also created with the *IOPE:placeholder* property set to the value 'Enter the new item(s) (separated by a comma)' in order to give users possibility to enter new instances.

3.4 Illustrative example (continued)

Figure 10 shows the IOPE_Web graph resulting from triggering the mapping rules that are applicable to the three constraint graphs displayed in Figures 2 and 3 for the focus class `PortACathPlacement`.

The blue color denotes the IOPE_Web sub-graph generated by triggering the first mapping rule (Figure 7) instantiated appropriately. The violet IOPE_Web part of the graph is the result of triggering the mapping rule in Figure 8 for the value restriction `(samsei:equipmentSupplies value samsei:sterile_compress)`, while the red part corresponds to the application of the mapping rule in Figure 9 for the cardinality restriction:
`(min 1 samsei:equipmentSupplies samsei:ProtectiveSupplies)`.
The green part corresponds to the application of a mapping rule given in Annex that is a variant of the mapping rule in Figure 9 when the class involved in the cardinality restriction has no sub-class and no instance in the domain ontology, like it is the case in the OntoSamsei ontology for the `samsei:VenousChest-SimulatorManikin` class in the cardinality restriction:
`(min 1 samsei:simulatorResources samsei:VenousChestSimulatorManikin)`

Figure 11 left shows the resulting pre-filled Web page generated by the HTML implementation of the IOPE_Web specification.

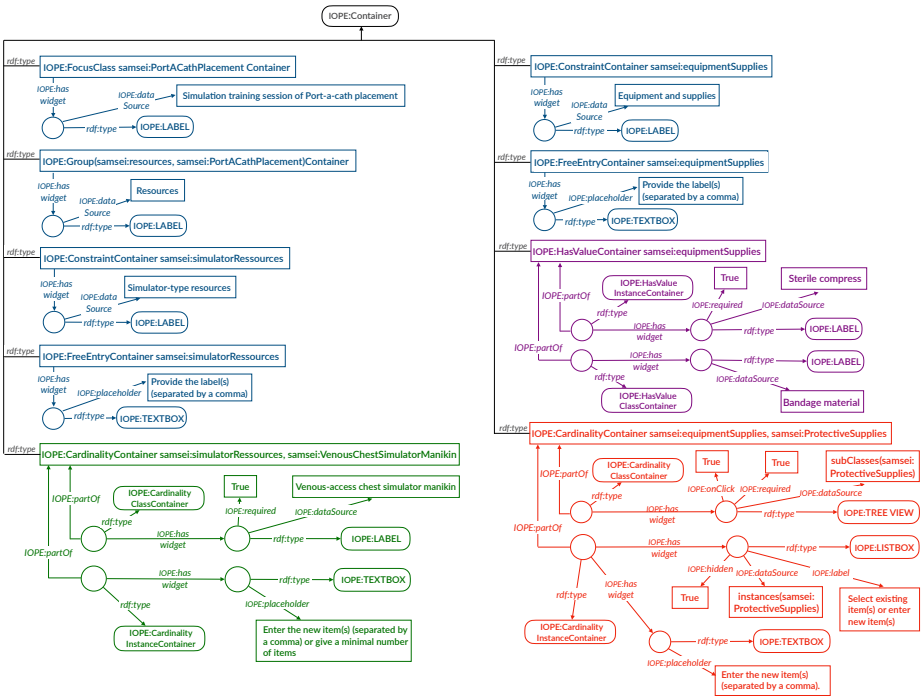

Fig. 10. Outcome of the application of mapping rules on three constraints examples (**samsei:simulatorResources min 1 samsei:VenousChestSimulatorManikin**), (**samsei:equipmentSupplies min 1 samsei:ProtectiveSupplies**) and (**samsei:equipmentSupplies value samsei:sterile_compress**).

Figure 11 right shows the effect of an user interaction through the widget of type the *IOPE:TREEVIEW* to select the sub-class *DisposableDrape* from the *ProtectiveSupplies* sub-class hierachy: the instance container corresponding to the selected sub-class becomes visible in order to allow the user to select an instance or to enter a new one.

The input entered by the user must be bound to RDF data corresponding to new instances or new constraints submitted to populate or enrich the domain ontology. This binding mechanism is based on a set of *binding rules* that are triggered on IOPE_Web graphs to generate RDF graphs built on the domain ontology.

3.5 Binding rules

Each binding rule is defined as a diagram with a IOPE_Web graph on the left side and the corresponding generated triples in the form of RDF graph on the right side.

The first binding rule shown in Figure 12 is triggered when a focus class F is chosen. This rule creates an instance f of the focus class F.

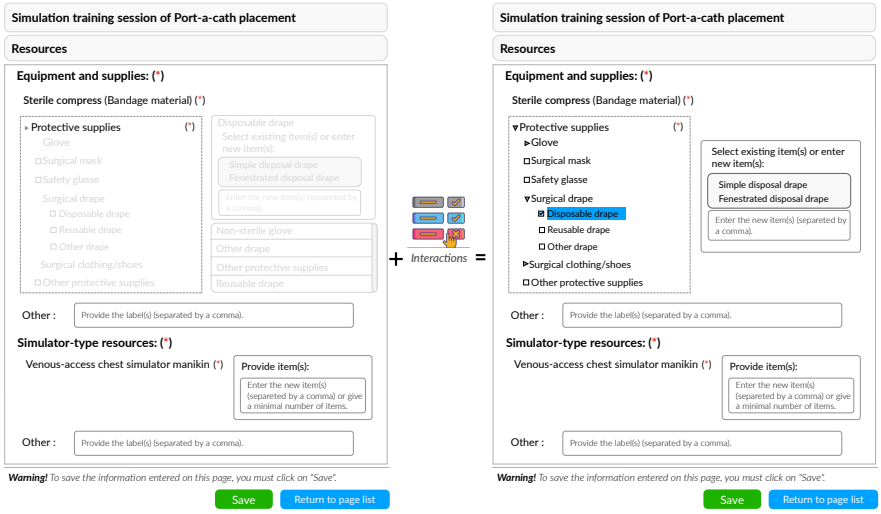

Fig. 11. Left: HTML Web page generated from the outcome of the application of mapping rules on three constraints of section 2.2. Right: HTML Web Page changes after user interaction by the widgets.

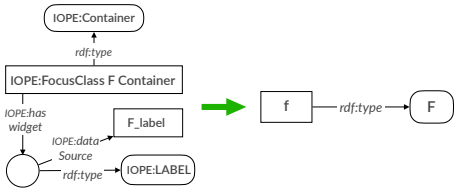

Fig. 12. Binding rule for a focus class F

The other binding rules are triggered when the `IOPE:value` property is filled by an input provided by the user through an interactive widget. Their application results in enriching the description of the instance f or in new constraints on the focus class F as follows.

Figure 13 shows the binding rule for the textbox widget in the free entry container of a property p for the focus class F. Its application generates a new constraint graph expressing a new cardinality constraint for F on the property p and a new class.

Figure 14 shows the binding rule for the chosen instance(s) from a listbox widget related to a selected class in treeview widget.

The other binding rules are listed in the Appendix.

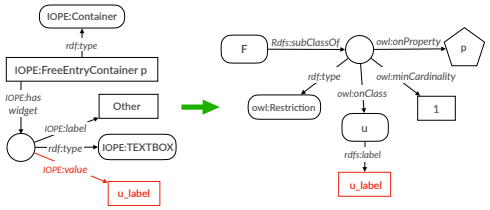

Fig. 13. Binding rule for free entry container on property p and a focus class F

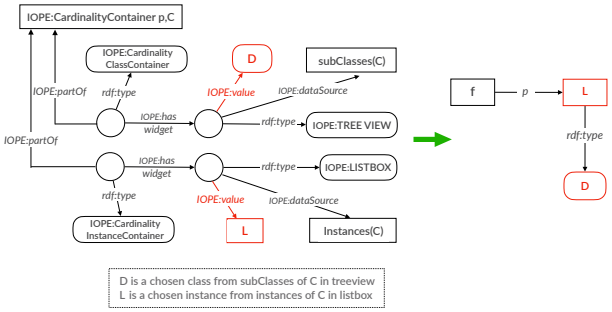

Fig. 14. Binding rule for a selected class D in a IOPE:TREEVIEW and selected instance(s) L from a IOPE:LISTBOX widget

4 Evaluation

The objective of our study is to evaluate the *efficiency*, the *users' satisfaction* and the *effectiveness* of the IOPE interface with the purpose of populating and enriching the OntoSAMSEI ontology.

The users involved in our user study are a subgroup of 22 experts in simulation-based training in Medicine among those who were solicited one year ago through an online questionnaire for bootstrapping the OntoSAMSEI ontology. They are domain experts but they are not familiar with RDF and OWL. The user study was organized in two steps for each expert. In the first step, the expert logs in the system with her credentials, picks one simulation training session, and begins to observe and update the information in the pre-filled Web pages. Once the expert is done with filling of the Web pages for one training session, she can choose another training session and follow the same process. In the second step, she will be transferred to a survey form to evaluate some qualitative aspects of IOPE and OntoSAMSEI ontology and reflect her viewpoint based on her interactions with the IOPE interface.

4.1 Evaluation of the IOPE GUI efficiency

We first provide quantitative results on the time spent by users and their number of interactions with IOPE . Then, we compare these results with the time insight

perceived by users and with the number of interactions required for the same tasks when interacting with a standard ontology editor.

Time spent by users and number of interactions. Each expert spent 163 seconds (2.72 minutes) on average, maximum 320 seconds (5.33 minutes), minimum 67 seconds (1.12 minutes). On average, their number of interactions with IOPE is 5.78, with a maximum of 14 and a minimum 3. The majority of interactions are with CHECK BOX widget (56.15%) followed by TEXT BOX widget (32.30%) and LIST BOX widget (11.53%).

Table 1 shows the distribution of experts in two categories of groups. In terms of number of interactions, we have built the groups of "prolific" experts (having more than 6 interactions with IOPE), "active" experts (having between 3 and 6 interactions), and "moderate" experts (with less than 3 interactions). In terms of interaction duration, we have built the groups of experts spending "short-time" (less than 2 minutes), "medium-time" (between 2 and 4 minutes), and "long-time" (more than 4 minutes).

Table 1. Distribution of expert groups

	moderate	**active**	**prolific**
Expert population	22.73%	50%	27.27%
	short-time	**medium-time**	**long-time**
Expert population	50%	31.82%	18.18%

The proportion of active experts, and also of short-time experts, is 50 %.

Table 2 reports the distribution of time groups for each interaction activity groups. We notice that more interactions do not necessary yield to more time spent to interact. Moreover, most prolific experts are short-time or medium-time, and most active and moderate experts are short-time. This shows that IOPE helps experts to fulfill their task in a reasonable amount of time, even for prolific experts.

Table 2. Distribution of interaction time groups for interaction volume groups.

		Interaction volume groups		
		moderate	**active**	**prolific**
Interaction time groups	**short-time**	0.80	0.46	0.33
	medium-time	0.00	0.27	0.67
	long-time	0.20	0.27	0.00

Time-to-insight users's evaluation. After they are done with using the IOPE interface for fulfilling their task, we ask the experts the following question to estimate the time-to-insight for a future interaction with IOPE : *"how much time do you expect to take for setting up a new simulation training session with* IOPE*?"*. The response is in the form of a Likert scale from 1 to 5 where "1" means "very short time" and "5" means "very long time".

Figure 15 shows the results.We observe that the majority of experts chose "short time" and "average time", i.e., options 2 and 3 in the Likert scale. Moreover, prolific experts and long-time perceive shorter expected time compared to the active and moderate experts. A possible interpretation is that more interactions and more time sent interacting with the system boosts the perception of faster delivery of required information, as the expert becomes more familiar with the different facets of information and their presentation through the interface. On the contrary, less interaction activity leads to a more scattered perception of expected time for future interaction, as shown with the moderare and short-time experts.

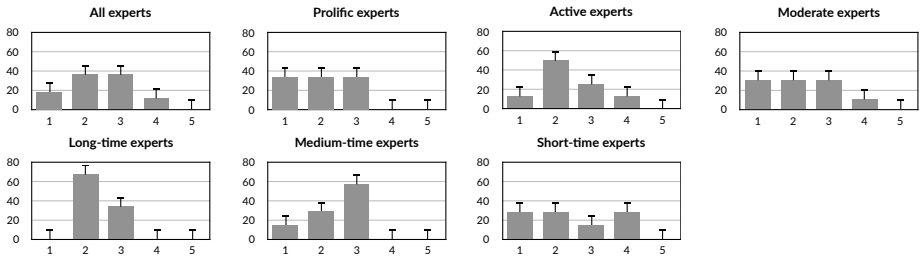

Fig. 15. Time-to-insight results.

Comparative efficiency of IOPE **with a standard ontology editor.** The goal of this experiment was to measure the added-value of IOPE compared to a standard ontology editor such as TopBraid [2], in terms of number of interactions required to fulfill edition tasks mentioned in Table 3. The tasks are categorized into three levels of difficulty based on the task-based taxonomy guidelines discussed in [8].

Table 3. Tasks descriptions.

Task	**Description** (Given the simulation training session X ...)
Easy	Fill the number of trainees for X
Medium	Fill the target audience of X
Difficult	Fill the required resources for X

For the fairness of this experiment, since none of the domain experts have ever used TopBraid, the different tasks were fulfilled by the five authors of the paper who have a sufficient knowledge about the domain, as well as a sufficient experience using both IOPE and TopBraid.

Figure 16 shows the average number of interaction steps to fulfil those tasks in IOPE and TopBraid.

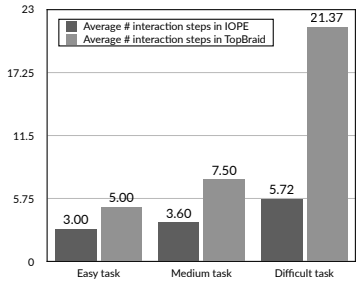

Fig. 16. Comparative number of interactions between IOPE and TopBraid.

We observe that for both tools, the number of interaction steps increases with the difficulty of the tasks. However, the IOPE's trend grows from average 3.00 steps for an easy task to average 5.72 steps for a difficult task, while using TopBraid grows from average 5.00 steps for an easy task to average 21.00 steps for a difficult task. This shows that IOPE , by weaving relevant information together using constraints, enables the experts to fulfill their tasks more rapidly than by using a standard editor. This is also in conformance with the time-to-insight experiment, as it depicts that IOPE is able to help experts achieve difficult tasks in a reasonable number of interaction steps.

4.2 Evaluation of IOPE users's satisfaction

We have measured on a Likert scale in the range 1 to 5 the assessment by users of three aspects of satisfaction, namely *utility*, *usability*, and *adoption*, through the questions of the three first rows of the Table 4. The aggregated results are shown in the three first column of Figure 17.

Utility. As depicted in the first column of Figure 17, 82.35% of the participants have a positive view on the utility of IOPE. However, the prolific experts appreciate the utility more than active experts, which concludes that more interactions blossom more utility. This shows that more interactions increases the perception of utility, which is also confirmed by long-time experts who are entirely on the positive spectrum.

Usability. As demonstrated in the second column of Figure 17, overall the experts perceived usability positively. However, there is a vivid contrast between moderate experts versus active and prolific experts, where the former group

Table 4. Measure definitions and corresponding questions asked in the survey.

Measures	Definition	Question asked in the survey
utility [21]	The usefulness of the method to fulfil a given task.	How do you evaluate the utility of IOPE for setting up simulation training sessions?
usability [17, 3]	The easiness of interactions with the method	To which degree do you find IOPE easy-to-use?
adoption [17]	The usefulness of the method for future similar tasks	How often will you employ IOPE for setting up and describing a new simulation training session in the future?
accuracy [16, 18]	The precision of information based on expert's prior knowledge.	How do you evaluate the accuracy of IOPE's pre-filled information for describing simulation training sessions?
completeness [17]	The retrieval exhaustiveness of the necessary and required information.	How do you evaluate the sufficiency of IOPE's pre-filled information for describing simulation training sessions?

seems to not enjoy the usability of IOPE. We conjecture that moderate experts got lost early in the process, and abandoned their task, hence their myopia on the overall usability. There is also a subset of long-time experts who assessed low usability. Given their session time of more than 4 minutes, our conjecture is that they spent time to fulfil their tasks more than needed, and got lost in the process also. .

Adoption. The choice over adoption is from 1 to 5, where 1 means "never" and 5 means "always". In general, the experts voted to adopt IOPE in the future most of the time.

4.3 Effectiveness of IOPE for enriching the OntoSAMSEI ontology

In this part of the experiment, we measure the experts assessment of *accuracy* and *completeness* of the OntoSAMSEI ontology through its presentation to the experts by IOPE GUI. We do it by asking to the experts the questions in the two last rows of the Table 4. The aggregated results (on the Likert scale from 1 to 5) are shown in the two last columns of Figure 17.

Accuracy. As depicted in the fourth column of Figure 17, the majority of the participants are positive on accuracy, while 11.76% are negative. Short-time and moderate experts express more negative votes on accuracy compared to long-time and prolific experts, respectively. This is presumably because less investigations in the former groups did not enable them a precise view of the ontology.

Completeness. As depicted in the last column of Figure 17, 76.46% of the participants find OntoSAMSEI complete enough. However, prolific experts appreciate completeness less than the overall population. By investigating the interactions of this group with the widgets, we found out that they prominently interact with text-boxes for enrichment. Hence this group has presumably more concerns of enriching and populating the ontology to make it more complete. The entire long-time expert group votes positively, which means that they put more time and effort to go into the details of the simulation training sessions and observe their completeness. The votes for medium-time and short-time experts are more scattered.

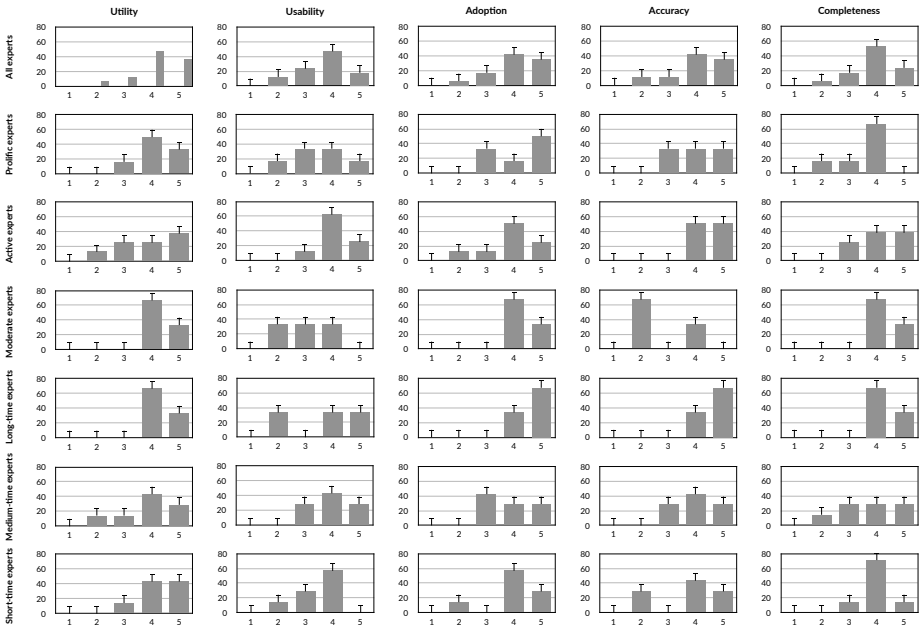

Fig. 17. Satisfaction and effectiveness metrics results.

5 Related work

A number of ontology editing tools have been proposed in the literature. The most widely used systems are Protégé [15] (and its Web version, WebProtégé [22]), TopBraid [2], and Ontodia [14], which offer means to create individuals based on ontologies. These systems require however a basic understanding of the RDF notation and of the OWL semantics to edit the ontology consistently. In contrast, with IOPE, all relevant information is presented and structured in an intuitive manner through Web forms. All RDF/OWL technicalities are hidden while the seamless enforcement of the ontological constraints is a strong guidance for users to update consistently the ontology.

WebVOWL [25] is a web application for the interactive graph-based visualization of ontologies which employs the Visual Notation for OWL Ontologies (VOWL) [11] as its infrastructure. WebVOWL defines graphical depictions for most elements of the Web Ontology Language (OWL). However, WebVOWL does not visualize the instances but only the OWL part of a (possibly populated) ontology. Also, the graphs displayed by the tool tend to become quickly illegible for domain expert when their size increases. In IOPE, we employ Web forms as a more widespread medium for visualizing information, and we support the update of instances and of ontological constraints.

Formulis [12] is a form-based user interface for updating RDF data. The method generates nested forms for relational aspects of knowledge graphs. The

experts are also guided with dynamic suggestions based on existing data. However, the nested structure makes the interface less intuitive than standard Web forms for increasing nesting depth. In addition, the approach is focused on the data part and does not consider OWL constraints. SCHÍMATOS [26] is another approach for creating and editing data in the form of shapes graph built using the SHACL standard[7]. VOCEDITOR [23] is an extension to reduce the introduction of erroneous data in the updates. While shapes graph seem well adapted for editing complex data, it requires the definition of such graphs for each ontology.

The closest approach to IOPE is ACTIVERAUL [7, 9], which automatically renders web forms from arbitrary ontologies. The resulting forms are instances of a User Interface (UI) ontology, called RaUL, and are generated by interpreting ontology assertions as rules. The UI ontology consists of elements describing the various form controls (textboxes, radio buttons, etc.) associated with the elements of the edited ontology. Inspired from ACTIVERAUL's UI ontology, IOPE stresses on ontological constraints as first-class citizens and renders pre-filled forms which provide a more aggregated view for the experts.

6 Conclusion

In this paper, we have presented the interactive IOPE framework for enrichment and population of specialized ontologies.Given any input ontology, IOPE exploits the ontological constraints and a set of mapping rules to generate a set of user-friendly Web pages which assist the experts in editing the ontology. Binding rules are then used to derive the RDF graphs corresponding to the updates entered by the experts. We have conducted an extensive set of experiments on the domain of simulation-based medical education, for measuring IOPE's efficiency, effectiveness, as well as the experts' satisfaction in fulfilling their tasks using IOPE . In the future, we plan to improve the *explainability* of IOPE to reduce the number of abandoned editing tasks and increase its usability by domain experts not familiar with ontology engineering.

References

1. IOPE implementation. `https://github.com/shadi-tabasi/IOPE.git`
2. Topquadrant topbraid composer. `https://www.topquadrant.com/products/topbraid-composer/`, accessed: 2021-01-15
3. Albert, W., Tullis, T.: Measuring the user experience: collecting, analyzing, and presenting usability metrics. Newnes (2013)
4. Allemang, D., Hendler, J.A.: Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL, Second Edition. Morgan Kaufmann (2011)
5. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: Dbpedia: A nucleus for a web of open data. In: ISWC. vol. 4825. Springer (2007)
6. Baghernezhad-Tabasi, S., Rousset, M.C., Druette, L., Jouanot, F., Meurger, C.: OntoSAMSEI: Interactive ontology modeling for supporting simulation-based training in Medicine. IC3K (2019), kEOD Doctoral Consortium

[7] Shapes Constraint Language (SHACL): https://www.w3.org/TR/shacl/

7. Butt, A., Haller, A., Liu, S., Xie, L.: Activeraul: Automatically generated web interfaces for creating rdf data. Semantic Web **2013** (2013)
8. Dimara, E., Franconeri, S., Plaisant, C., Bezerianos, A., Dragicevic, P.: A task-based taxonomy of cognitive biases for information visualization. IEEE Trans. Vis. Comput. Graph. **26**, 1413–1432 (2020)
9. Haller, A., Umbrich, J., Hausenblas, M.: Raul: Rdfa user interface language - A data processing model for web applications. In: WISE. vol. 6488, pp. 400–410. Springer (2010)
10. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers (2011)
11. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with VOWL. Semantic Web **7**(4), 399–419 (2016)
12. Maillot, P., Ferré, S., Cellier, P., Ducassé, M., Partouche, F.: Nested forms with dynamic suggestions for quality RDF authoring. In: DEXA. vol. 10438, pp. 35–45. Springer (2017)
13. Matuszek, C., Cabral, J., Witbrock, M.J., DeOliveira, J.: An introduction to the syntax and content of cyc. In: AAAI Spring Symposium. pp. 44–49 (2006)
14. Mouromtsev, D., Pavlov, D., Emelyanov, Y., Morozov, A., Razdyakonov, D., Galkin, M.: The simple web-based tool for visualization and sharing of semantic data and ontologies. In: International Semantic Web Conference (2015)
15. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with protégé-2000. IEEE Intell. Syst. **16**(2), 60–71 (2001)
16. Omidvar-Tehrani, B., Amer-Yahia, S.: Data pipelines for user group analytics. In: SIGMOD Conference. pp. 2048–2053. ACM (2019)
17. Rahman, P., Jiang, L., Nandi, A.: Evaluating interactive data systems. VLDB J. **29**(1), 119–146 (2020)
18. Rahman, P., Jiang, L., Nandi, A.: Evaluating interactive data systems. The VLDB Journal **29**(1), 119–146 (2020)
19. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: 16th international conference on World Wide Web. pp. 697–706. ACM (2007)
20. Tanon, T.P., Vrandecic, D., Schaffert, S., Steiner, T., Pintscher, L.: From freebase to wikidata: The great migration. In: WWW. pp. 1419–1428. ACM (2016)
21. Thomas, J.J.: Illuminating the Path: The Research and Development Agenda for Visual Analytics (2005)
22. Tudorache, T., Nyulas, C., Noy, N.F., Musen, M.A.: Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. Semantic Web **4**(1), 89–99 (2013)
23. Valdestilhas, A., Publio, G., Cimmino Arriaga, A., Riechert, T.: Voceditor -an integrated environment to visually edit, validate and versioning rdf vocabularies (12 2020). https://doi.org/10.13140/RG.2.2.12192.81921
24. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)
25. Wiens, V., Lohmann, S., Auer, S.: Webvowl editor: Device-independent visual ontology modeling. In: ISWC 2018 Posters & Demonstrations. CEUR Workshop Proceedings, vol. 2180
26. Wright, J., Méndez, S.J.R., Haller, A., Taylor, K., Omran, P.G.: Schímatos: A shacl-based web-form generator for knowledge graph editing. In: ISWC. vol. 12507, pp. 65–80. Springer (2020)

A Appendix

A.1 Mapping rules

Mapping rule of domain, range constraints for a property `(p)` such that, `(p rdfs:domain F) (p rdfs:range C)`. The mapping rule is described in Figure 18 when the class range of the property p has subClasses and instances. If the class range of the property p has subClasses and instances, the mapping rule is described in Figure 18. If the class range of the property p is without subClasses and instances, the mapping rule is described in Figure 19

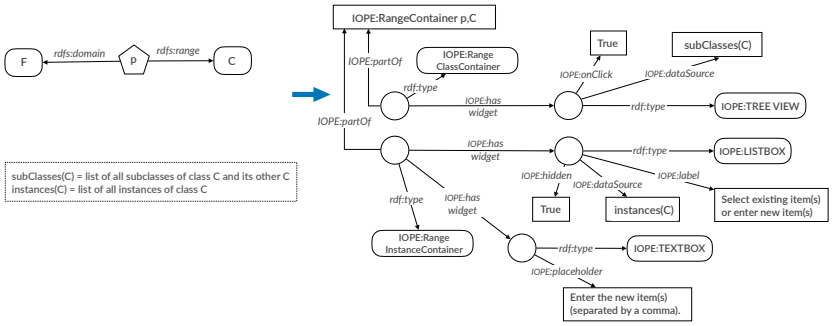

Fig. 18. Mapping rule for domain, range constraints where subClasses(C) and instance(C) are not empty

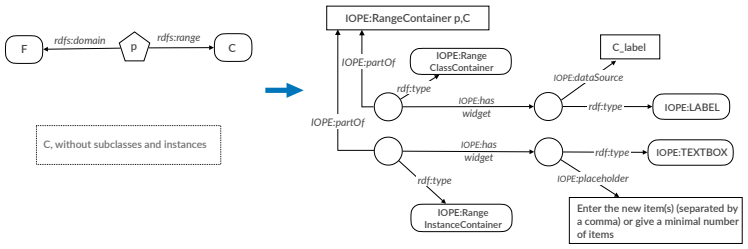

Fig. 19. Mapping rule for domain, range constraints where subClasses(C) and instance(C) are empty

Mapping rule for a cardinality restriction `(p min n C)` such that $n > 0$, `C` does not have hierarchy of sub-classes nor list of instances in the domain ontology. This mapping rule is described in Figure 20. For the *IOPE:CardinalityClassContainer*, a widget of type *IOPE:LABEL* is created as a blank node with the property *IOPE:dataSource* filled by the the label of class C, which is C_label. The property *IOPE:required* is set to 'True' for this widget

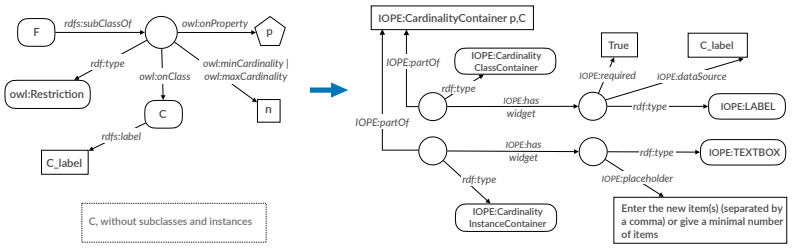

Fig. 20. Mapping rule for a Cardinality constraint where subClasses(C) and instance(C) are empty

to indicate that this value is mandatory for the property p. As class C does not have any instances for the IOPE:CardinalityInstanceContainer, a widget of type *IOPE:TEXTBOX* is created with the *IOPE:placeholder* property set to the value 'Enter the new item(s) (separated by a comma) or give a minimal number of items' in order to give users possibility to enter new instances or give a minimum number that (s)he needs.

Mapping rule for a cardinality restriction (`p min n C`) such that $n > 0$, `C` does not have hierarchy of sub-classes but has a list of instances in the domain ontology. This mapping rule is described in Figure 21. For the *IOPE:CardinalityClassContainer*, a widget of type *IOPE:LABEL* is created as a blank node with the property *IOPE:dataSource* filled by the the label of class C, which is C_label. The property *IOPE:required* is set to 'True' for this widget to indicate that this value is mandatory for the property p. For the IOPE:CardinalityInstanceContainer, a widget of type *IOPE:LISTBOX* is created as a blank node with the property *IOPE:dataSource* filled by the list $instances(C)$ of instances of C , the *IOPE:label* property set to 'select existing item(s) or enter new item(s)' and the *IOPE:hidden* property set to False to make the widget visible and intractable to the user. A widget of type *IOPE:TEXTBOX* is also created with the *IOPE:placeholder* property set to the value 'Enter the new item(s) (separated by a comma)' in order to give users possibility to enter new instances.

Mapping rule for a cardinality restriction (`p min n C`) such that $n > 0$, `C` has a hierarchy of sub-classes but does not have list of instances in the domain ontology. This mapping rule is described in Figure 22. For the *IOPE:CardinalityClassContainer*, a widget of type *IOPE:TREEVIEW* is created as a blank node with the property *IOPE:dataSource* filled by the the tree view of $subClasses(C)$, which denotes the hierarchy of the sub-classes of C in the domain ontology enriched with an additional item $Other\_C$. The property *IOPE:required* and *IOPE:onClick* are set to 'True' for this widget to indicate that entering at least one value is mandatory for the property p and that this widget supports the interaction with users to display interactively the sub-class hierarchy. For the *IOPE:CardinalityInstanceContainer*, a widget of type *IOPE:TEXTBOX* is created with the *IOPE:placeholder* property set to

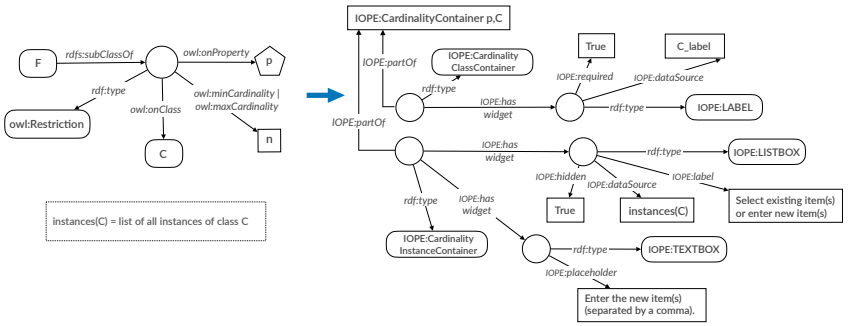

Fig. 21. Mapping rule for a Cardinality constraint where subClasses(C) and instance(C) are empty

the value 'Enter the new item(s) (separated by a comma) or give a minimal number of items' in order to give users possibility to enter new instances or give a minimum number that (s)he needs.

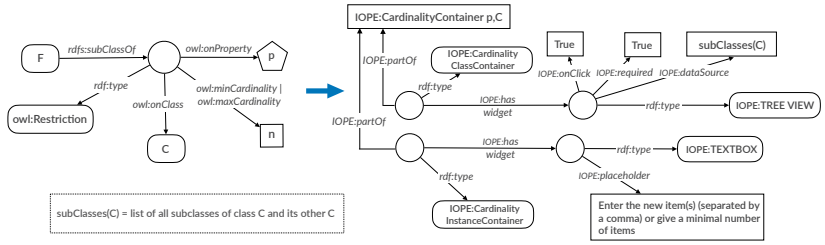

Fig. 22. Mapping rule for a Cardinality constraint where subClasses(C) and instance(C) are empty

Mapping rule for a cardinality restriction (`p min n`) such that $n > 0$, `p` is a datatype property. This mapping rule is described in Figure 23.

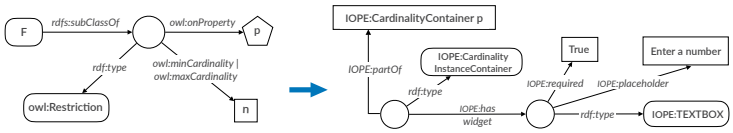

Fig. 23. Mapping rule for a Cardinality constraint on a datatype property p.

For the *IOPE:CardinalityInstanceContainer*, a widget of type *IOPE:TEXTBOX* is created with the *IOPE:placeholder* property set to the value 'Enter a number' in order to give users possibility to enter the numerical value (s)he needs for the

property p. Figure 24 shows the binding rule for a textbox widget filled by a numerical value related to the datatype property p.

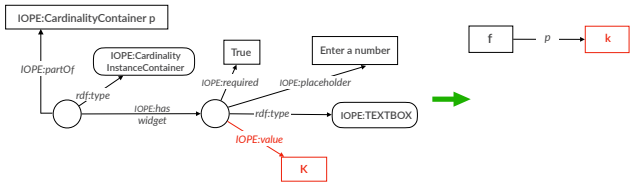

Fig. 24. Binding rule for a value of type integer in a IOPE:TEXTBOX widget related to the datatype property p.

Mapping rule for a rdf container i.e., rdf:Bag, rdf:Seq, rdf:Alt. This mapping rule is described in Figure 25.

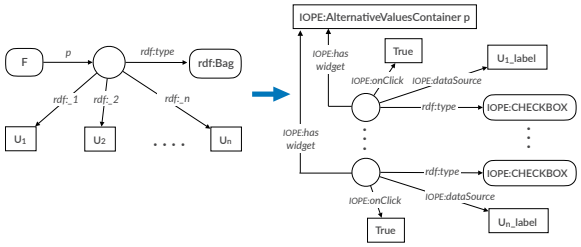

Fig. 25. Mapping rule for a rdf container i.e., rdf:Bag, rdf:Seq, rdf:Alt.

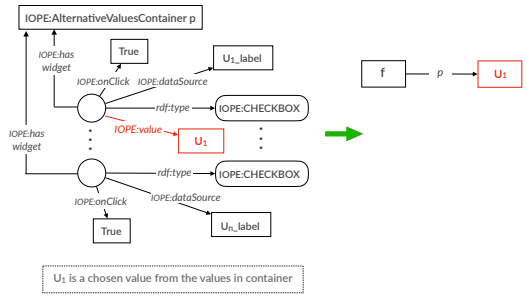

Fig. 26. Binding rule for a selected value U_1 via a IOPE:CHECKBOX widget within the values in rdf containe.

A.2 Binding rules

Figure 27 shows the binding rule for a textbox widget filled by a value of type string related to a selected class in a treeview widget.

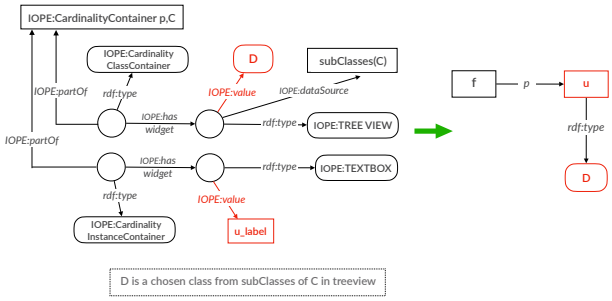

Fig. 27. Binding rule for a selected class D in a IOPE:TREEVIEW and its entered item(s) in a IOPE:TEXTBOX widget

Figure 28 shows the binding rule for a textbox widget filled by a value of type string related to a label widget.

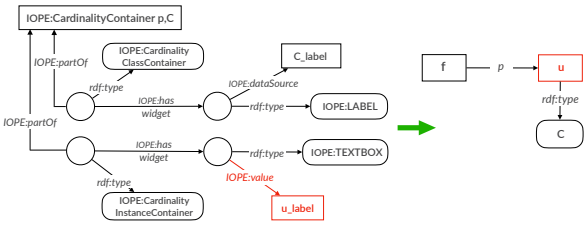

Fig. 28. Binding rule for value of type string in a IOPE:TEXTBOX widget related to a label widget

Figure 29 shows the binding rule for a text-box widget filled by a value of type integer related to a label widget.

Figure 30 shows the binding rule for a textbox widget filled by a value of type string related to a `Other_C` in a treeview widget.

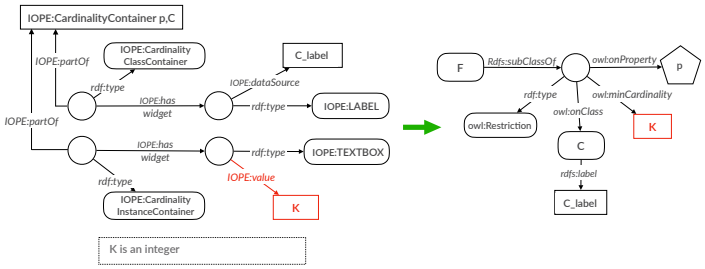

Fig. 29. Binding rule for a value of type integer in a IOPE:TEXTBOX widget related to a label widget

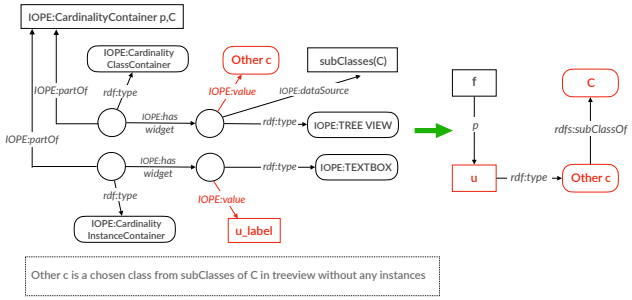

Fig. 30. Binding rule for a text-box widget filled by a value of type string related to a class `Other_C`