IOPE: Interactive Ontology Population and Enrichment Guided by Ontological Constraints

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

[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. Specialized ontologies are constructed to capture the skills of experienced experts, with the goal of sharing them with a larger community of trainees and less experienced experts in the domain. A challenging task in the engineering pipeline of specialized ontologies is *ontology updating*, which encompasses enrichment and population. Domain experts require an understanding of the RDF notation and OWL semantics to update their ontologies, which is not often the case. In this paper, we present IOPE, an interactive framework for the automatic construction of a Graphical User Interface (GUI) to support the controlled update process, by enabling the experts to easily interact with their ontology. We contribute a set of "mapping rules" to transform the ontological constraints into interactive widgets organized in the form of pre-filled Web pages in the GUI, and a set of "binding rules" to transform the expert interactions into RDF graphs, and hence perform the updates. In an extensive set of experiments, we illustrate the efficacy of IOPE in empowering medical experts to update their specialized ontology.

1 Introduction

Ontologies are the backbone of many information systems that require access to structured knowledge. By their very nature, real world ontologies are dynamic artifacts that evolve both in their structure (i.e., the data model) and their content (i.e., instances). Keeping them up-to-date is a critical operation for most applications which rely on semantic Web technologies. Ontology updates encompass both *enrichment* and *population*. 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, through domain documentations. Ontology updates are typically performed in an exploratory and manual fashion, as the non-documented knowledge of the domain expert is required to be taken into consideration. However, these manual updates put burden on the experts and render the whole ontological ecosystem inefficient. In this paper, we advocate for an alternative and more effective approach, and propose to handle updates automatically through a few interactions with the expert, using a Graphical User Interface (GUI).

The challenges associated to interaction-based automatic updates are two-fold: (*i*) While ontologies are typically represented in the form of graphs, it is inherently difficult and counterintuitive to provide a graphical graph-based representation of ontologies for the consumption of experts. While there exist several methods to visualize a graph structure [10, 8], the outcome is often hard to digest by domain experts. (*ii*) It is unclear how experts should perform ontology updates through the interactions, without the prior knowledge of the formal syntax and the semantics of ontology languages.

In this paper, we demonstrate IOPE (Interactive Ontology Population and Enrichment), a framework for the automatic construction of a Graphical User Interface (GUI) using *prefilled Web forms*. We leverage Web forms as a natural interaction means to tackle the challenge of counterintuitive ontology representations. IOPE generates and prefills the Web forms from *ontological constraints*, which support the controlled update process of a given ontology. While IOPE is generic and can be applied to ontologies from a variety of domains, we employ an ontology called OntoSAMSEI [4] as a use case, whose content helps the domain experts design teaching units for learning skills in simulation-based Medicine. OntoSAMSEI is a hierarchy of classes and properties enriched by ontological constraints on those classes and properties, that convey the constraints that will have to be fulfilled by their future sub-classes, sub-properties or instances. We show how to exploit such ontological constraints as a source of guidance for (possibly less experienced) educators willing to design their own simulation sessions, hence addressing the challenge of expert noviceship. In [3], we present practical examples of the application of IOPE on OntoSAMSEI. Moreover, OntoSAMSEI's IOPE GUI is accessible via the following link (in French): **http://iope.tabasi.info**.

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 from 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 domain of interest based on a structured vocabulary made of classes, properties and instances. Ontological constraints are 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

Table 1. RDFS and OWL constraints considered in this paper

Type	**Shortened syntax**	**Semantics**
Class specialization	`(C rdfs:subClassOf D)`	$\forall$ i ((i rdf:type C) $\Rightarrow$ (i rdf:type D))
Property specialization	`(p rdfs:subPropertyOf q)`	$\forall$ i $\forall$ j ((i p j) $\Rightarrow$ (i q j))
Domain restriction	`(p rdfs:domain C)`	$\forall$ i $\forall$ j ((i p j) $\Rightarrow$ (i rdf:type C))
Range restriction	`(p rdfs:range D)`	$\forall$ i $\forall$ j ((i p j) $\Rightarrow$ (j rdf:type D))
Value restriction	`(C p owl:hasValue v)`	$\forall$ i ((i rdf:type C) $\Rightarrow$ (i p v))
Alternative values restriction	`(C p owl:oneOf [v1, ..., vn])`	$\forall$ i ((i rdf:type C) $\Rightarrow \bigvee_{k\in[1..n]}$ (i p vk))
Cardinality restriction	`(C p min k D)`	$\forall$ i ((i rdf:type C) $\Rightarrow \exists o_1, \dots o_k(\bigwedge_{i,j\in[1..k]} o_i \neq o_j$ $\wedge \bigwedge_{j\in[1..k]}$ (o_j rdf:type D) $\wedge$ (i p o_j))

specific values for some properties. The ontological constraints are defined in RDFS[4] and OWL[5], and described as RDF graphs.

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) 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 (a.k.a. a knowledge graph) is a set of (factual or ontological) statements expressed as triples $(s, p, o) \in (I \cup B) \times I \times (I \cup L \cup B)$ that form a graph whose nodes are IRIs, blank nodes or literals.

2.2 Ontological Constraints

The ontological constraints that we consider are RDFS constraints and some OWL constraints (displayed in Table 1). Figure 1 visualize the RDF graphs associated to two constraints declared in the OntoSAMSEI 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 port or a catheter. The RDF graph in Figure 1(a) expresses that `samsei:sterile_compress` (which is an instance of Bandage material) is declared in the ontology as a mandatory value of the property `samsei: equipmentSupplies`. The RDF graph depicted in Figure 1(b) expresses that at least one equipment of type `samsei:protectiveSupplies` is mandatory for simulating a placement of a port or a catheter.

3 Interactive Ontology Update

Our approach consists of transposing the RDF data and the ontological constraints of a given domain ontology into a graphical user interface (GUI) named

[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`

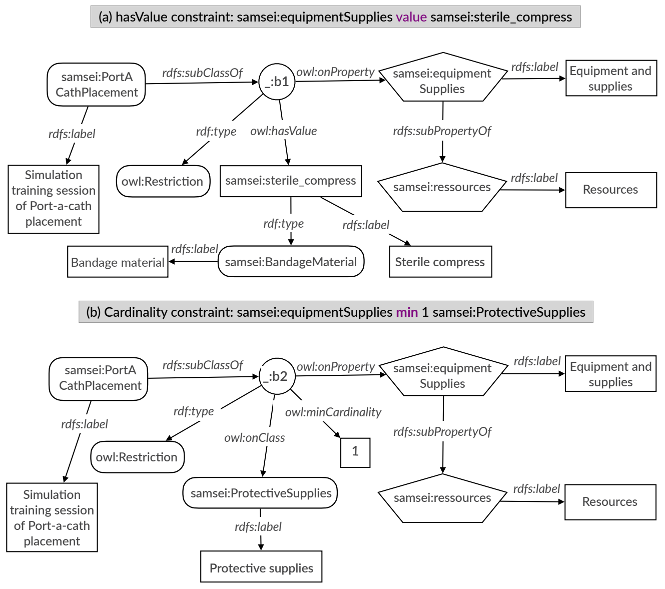

Fig. 1. Two constraint graphs (a) and (b) on the property `equipmentSupplies` for the class `PortACathPlacement`

IOPE GUI (step 1 in Figure 2). It functions as a guidance for domain experts to easily explore the ontology (step 2) and update it (step 3) through interactive graphical widgets. The input entered by domain experts through the IOPE GUI are then transformed into RDF triples (step 4) that must be verified by an ontology engineer (step 5) before being permanently added in the domain ontology (step 6). Figure 2 provides an overview of IOPE's workflow.

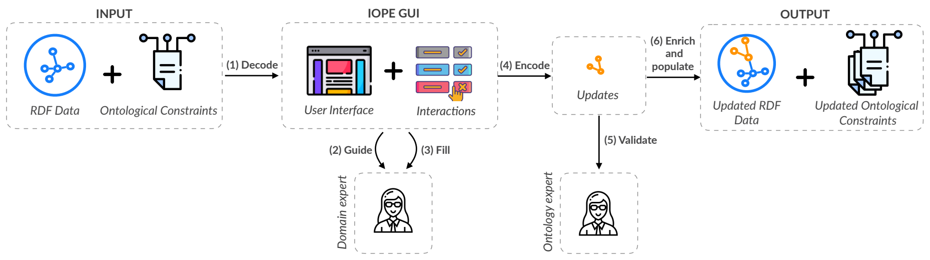

Fig. 2. The overview of IOPE's workflow.

The *Web form templates* on which the IOPE GUI is built are described using a Web form ontology called IOPEWeb that we have developed by adapting RaUL [9].

3.1 The IOPEWeb Ontology

The IOPEWeb ontology is shown in Figure 3. It is organized around 4 main classes, i.e., `IOPE:Page`, `IOPE:PageLayout`, `IOPE:Container` and `IOPE:Widget`. These classes are related by properties for modeling Web pages. The Web pages themselves are structured in the form of containers filled with widgets. Each Web page is also associated to a page layout.

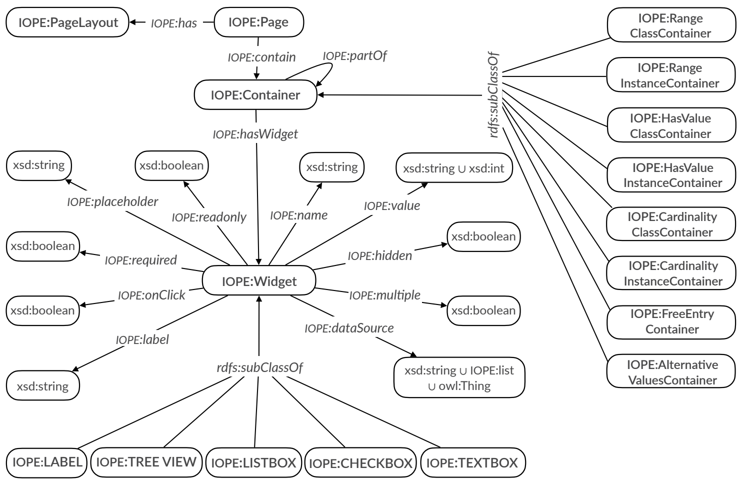

Fig. 3. The IOPE Web form ontology

The widgets are the direct point of user interaction, which are associated to the underlying RDF graph for the input ontology. The visualization and the user interaction are done using several types of widgets, such as label, tree view, list box, text box, and check box. These widgets constitute the subclasses of the main class `IOPE:Widget`, and inherit the standard widget properties described in IOPEWeb.

IOPEWeb describes how the input and output of widgets are modeled. We employ the `IOPE:dataSource` property for the assignment of an input data (from a domain ontology) of type `xsd:string`, simple or nested list `IOPE:list`, or `owl:Thing`, to their corresponding widgets. The `IOPE:value` property is filled by the value, entered by the user through the widget.

Widgets can be grouped in a Web page within containers. The containers themselves can be nested using the `IOPE:partOf` property. In our setting, different types of specific containers are considered as subclasses of `IOPE:Container` to express that the different types of ontological constraints will be rendered differently in IOPE GUI.

3.2 Ontology-Based GUI Construction

In a declarative approach, we employ a set of *mapping rules* to generate automatically pre-filled Web pages, and a set of *binding rules* to generate RDF graphs from the values entered by users through the widgets.

Input: The input required for GUI construction is a domain ontology in which the ontological constraints are automatically saturated by a reasoning algorithm as detailed in [5].

Initialization: The GUI construction is initiated with the selection of one class of interest in the ontology by the user, called the *focus class* F. The set $Constraints(F)$ of the ontological constraints associated to F is decomposed in groups $Group(P, F)$ of all the constraints involving sub-properties of a given property P. 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 4. The page layout defines the organization of the Web page with a set of specific containers dedicated to different ontological constraints on sub-properties p of P for which there exists constraints in $Group(P, F)$.

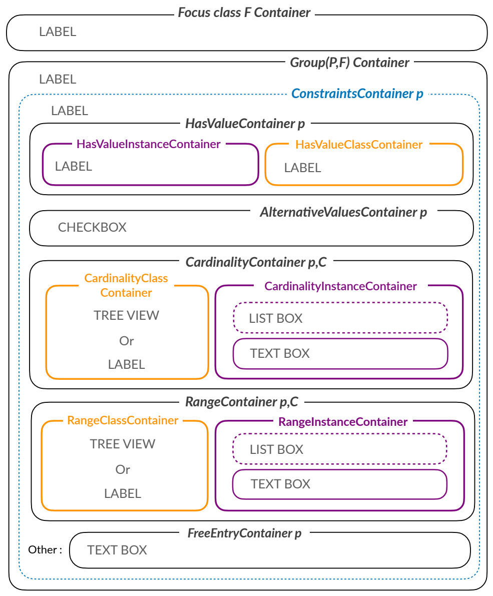

Fig. 4. Web page template prepared for the rendering of constraints of the focus class F for each property p which is a specialization of a same property P.

The following instances of the `IOPE:Container` class are created, with their pre-allocated positions in the empty Web page template shown in Figure 4.

– "`IOPE:FocusClass` F `Container`" denotes the main container of the created Web page for the focus class F;
– "`IOPE:`$Group(P, F)$ `Container`" denotes the container that groups all the other containers corresponding to the constraints holding for the class F on the sub-properties of P;
– "`IOPE:ConstraintContainer` p" denotes the container which contains restrictions of F on the property p, where p is a sub-property of P;
– "`IOPE:HasValueContainer` p" denotes the container which contains the `HasValue` restrictions of F on the property p;
– "`IOPE:AlternativeValuesContainer` p" denotes the container which contains the `AlternativeValues` restrictions of F on the property p;
– "`IOPE:CardinalityContainter` p, C" denotes the container which contains the `cardinality` restrictions of F on the property p and the class C;
– "`IOPE:RangeContainter` p, C" denotes the container which contains the `range` restrictions of F on the property p, where the range of p is the class C;
– "`IOPE:FreeEntryContainer` p" denotes the container which enables the user to add new classes involved in the cardinality restrictions for the property p.

Then, the mapping rules are triggered to map components of each ontological constraint to the widgets inside the aforementioned containers, and fill each Web page guided by the ontology.

Mapping rules: Each mapping rule has a constraint graph pattern in its left-hand side, and an IOPEWeb graph pattern in its right-hand side. The constraint graph pattern expresses a particular ontological constraint on a property and a (focus) class, and the IOPEWeb graph pattern specifies how to pre-fill the corresponding container to render this ontological constraint. Each rule is instantiated by mapping the constraint graph pattern in its left-hand side to the constraints graphs present in the input ontology and involving the chosen focus class. The mapping rules can be triggered in a forward-chaining manner and in any order. The resulting IOPEWeb graph provides the full RDF specification of the pre-filled Web pages that have to be created for the focus class F chosen by the user. We provide the exhaustive set of mapping rules in [5]. In this paper though, we just give two of them in their instantiated form for clarity purpose.

Figure 5 shows a mapping rule for a value restriction `(`F p `value` v`)`. The specific container "`IOPE:HasValueContainer` p" is decomposed into two sub-containers defined as blank nodes, whose types are `IOPE:HasValueInstanceContainer` and `IOPE:HasValueClassContainer`. For the 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 from the domain ontology. The property `IOPE:required` is set to `True` for the first widget, to show that the value v is mandatory for the property p.

Figure 6 shows a mapping rule for a cardinality restriction `(`F p `min` n C`)` such that $n > 0$, where C has a hierarchy of sub-classes and a list of instances in the domain ontology. The specific container "`IOPE:CardinalyContainer` p, C"

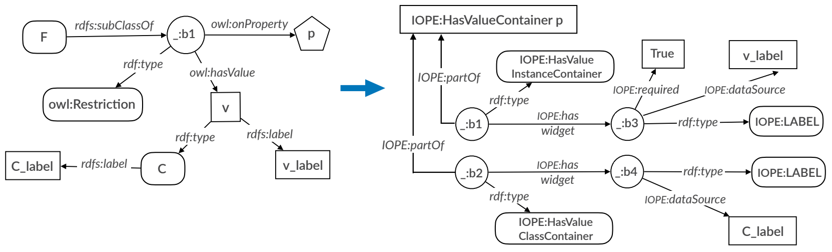

Fig. 5. Mapping rule for a value restriction (F p `value` v) where v `rdf:type` C

is decomposed into two sub-containers defined as blank nodes, with types "`IOPE:CardinalityClassContainer`" and "`IOPE:CardinalityInstanceContainer`".

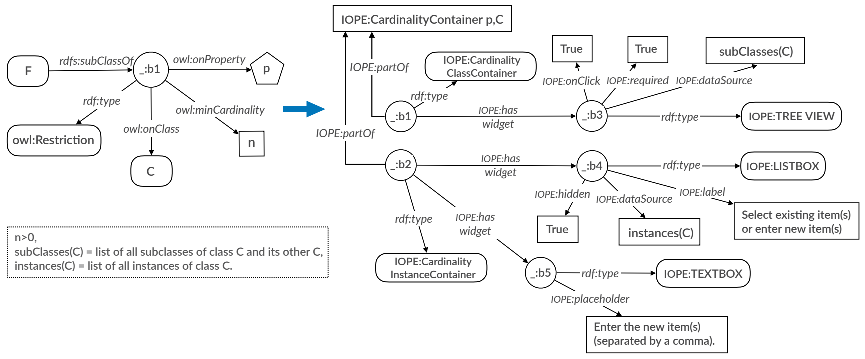

Fig. 6. Mapping rule for a Cardinality constraint where $subClasses(C)$ and $instances(C)$ are not empty, and $n > 0$.

For the first sub-container, a widget of type `IOPE:TREEVIEW` is created as a blank node with the property `IOPE:dataSource` filled by the tree view of $subClasses(C)$, i.e., the hierarchy of C's sub-classes in the domain ontology, enriched with an additional item `Other_C`. The property `IOPE:required` and `IOPE:onClick` are also set to True for this widget to indicate that (i) entering at least one value is mandatory for the property p, and (ii) this widget supports the interaction with users to display the sub-class hierarchy, interactively.

For the second sub-container, 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 the class C. The `IOPE:label` property is set to "select existing item(s) or enter new item(s)" and the `IOPE:hidden` property is 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, whose value is set to "enter the new item(s) (separated by a comma)", in order to enable the user to enter new instances.

The input entered through user interactions must then 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* which are triggered on the IOPEWeb graph to generate RDF graphs. Next, we will discuss these binding rules.

Binding rules: The role of binding rules is to transform user interactions into RDF graphs. A binding rule has an IOPEWeb graph pattern in its left-hand side, and a RDF graph pattern in its right-hand side. The binding rule is triggered when an input is entered by a user in a IOPEWeb form instantiating the left-hand side of the binding rule. The corresponding instantiation of the right-hand side provides RDF triples that have to be added in the output RDF graph. There are 9 binding rules. In this paper, we describe 2 of them, and provide the rest in [5]. The binding rule shown in Figure 7 is triggered when a focus class F is chosen by the user. Once triggered, the rule creates an instance *new_f* of the focus class F in the output RDF graph.

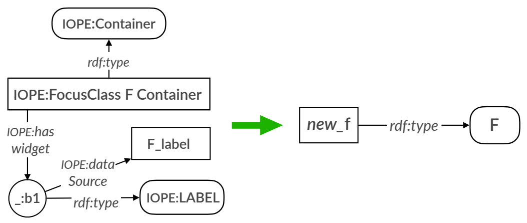

Fig. 7. for creating an instance *new_f* of a focus class F.

The other binding rules are triggered when the `IOPE:value` property is filled by an input provided by the user through an interactive widget. Figure 8 shows the binding rule for the `IOPE:TEXTBOX` widget in the free entry container of a property p for the focus class F. Once this binding rule is triggered, a new constraint graph will be generated which expresses a new class and a new `cardinality` constraint for F on the property p.

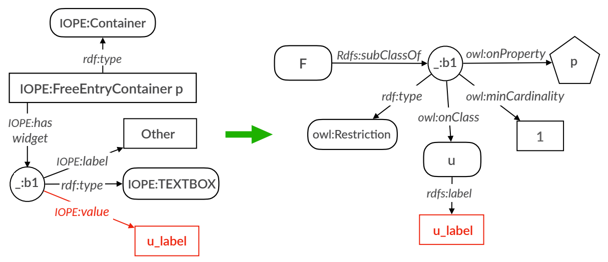

Fig. 8. Binding rule for free entry container on property p and a focus class F

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. 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. 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

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 2 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 3 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. This shows that IOPE helps experts to fulfill their task in a reasonable amount of time, even for prolific experts.

Table 2. Distribution of expert groups

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

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".

Table 3. 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

Figure 9 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.

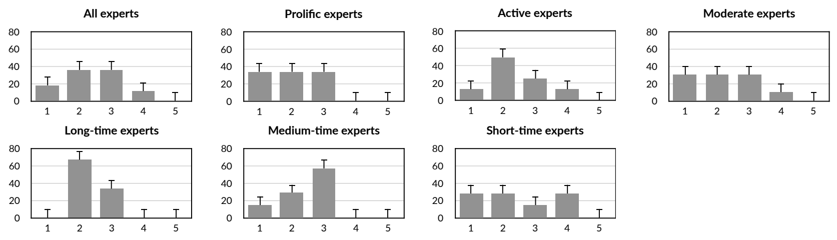

Fig. 9. Prediction of experts about time-to-insight for their next utilization of the IOPEinterface. Dashed bars show the average values of time-to-insights for all the experts.

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 [1], in terms of number of interactions required to fulfill edition tasks mentioned in Table 4.1. The tasks are categorized into three levels of difficulty based on [7].

Table 4. 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 10 shows the average number of interaction steps to fulfil those tasks in IOPE and TopBraid.

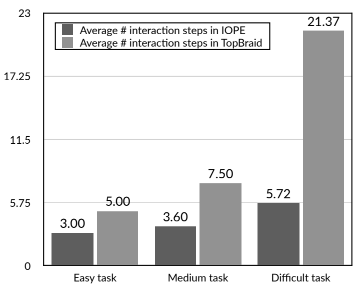

Fig. 10. Average number of interactions in IOPE and TopBraid.

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.

4.2 Evaluation of IOPE Users' 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 Table 5.

Table 5. Measure definitions and corresponding questions asked in the survey.

Measures	Definition	Question asked in the survey
utility [16, 2]	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 [15, 2]	The easiness of interactions with the method	To which degree do you find IOPE easy-to-use?
adoption [16]	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 [14, 15]	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 [15]	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?

The aggregated results are shown in the three first column of Figure 11.

Utility. 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. 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.

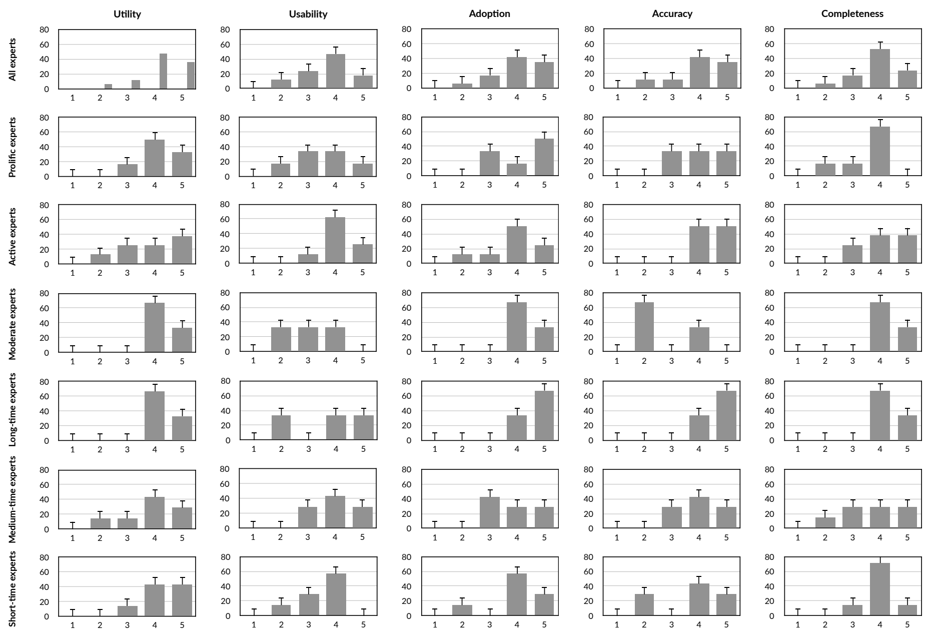

Fig. 11. Satisfaction and effectiveness metrics results.

Usability. Overall the experts perceived usability positively. However, there is a vivid contrast between moderate experts versus active and prolific experts, where the former group seems to not enjoy the usability of IOPE. We conjecture that moderate experts got lost early in the process, and abandoned their task. There is also a subset of long-time experts who assessed low usability. They probably spent too much time to fulfil their tasks 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". Most of the experts voted to adopt IOPE in the future.

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 the experts the questions in the two last rows of the Table 5. The aggregated results (on the Likert scale from 1 to 5) are shown in the two last columns of Figure 11.

Accuracy. 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 fewer investigations in the former groups did not enable them a precise view of the ontology.

Completeness. 76.46% of the participants find OntoSAMSEI complete enough. However, prolific experts appreciate completeness less than the overall population. We found out that they prominently interact with text-boxes, which shows that they use IOPE to effectively enrich the ontology. The entire long-time expert group votes positively, which means that spending more time to go into the details of the simulation training sessions convinces them of their completeness.

5 Related Work

In the literature, ontological updates are often performed using ontology editing tools [13, 1]. However, these systems require a basic understanding of the RDF notation and of the OWL semantics to edit the ontology consistently. Graph-based editing approaches alleviate this limitation by leveraging shapes graphs in the form of SHACL standard[6] [19, 17]. While shapes graphs are well adapted for editing complex data, they require the definition of such graphs for each ontology. In contrast, IOPE abstracts all RDF/OWL technicalities and seamlessly enforces the ontological constraints as a strong guidance for the experts to update the ontology, using the pre-filled forms.

WebVOWL [18] is a web application for the interactive graph-based visualization of ontologies which employs the Visual Notation for OWL Ontologies (VOWL) [11]. 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 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.

Forms are also used in [12] in a nested structure to capture relational aspects of knowledge graphs and update RDF data. However, the nested structure introduces increasing complexity and hence lacks intuitiveness. Moreover, the focus in [12] is solely on the population part and the approach does not extend to OWL constraints. In [6], Web forms are generated from ontologies (using a User Interface ontology, called RaUL) by interpreting ontology assertions as rules. While the approach only incorporates individual assertions (ontology population), IOPE serves both ontology enrichment and population, through interactions with the experts. IOPE stresses on ontological constraints as first-class citizens and renders pre-filled forms to provide a more aggregated view for the experts, which is, to the best of our knowledge, nonexistent in the literature.

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

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

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. Topquadrant topbraid composer. `https://www.topquadrant.com/products/topbraid-composer/`, accessed: 2021-01-15
2. Albert, W., Tullis, T.: Measuring the user experience: collecting, analyzing, and presenting usability metrics. Newnes (2013)
3. Baghernezhad-Tabasi, S., Druette, L., Jouanot, F., Meurger, C., Rousset, M.C.: IOPE: Interactive Ontology Population and Enrichment. In: SEMANTiCS (2021)
4. Baghernezhad-Tabasi, S., Druette, L., Jouanot, F., Meurger, C., Rousset, M.C.: OntoSAMSEI: Interactive ontology engineering for supporting simulation-based training in medicine. In: WETICE (2021)
5. Baghernezhad-Tabasi, S., Rousset, M.C., Druette, L., Jouanot, F., Meurger, C.: IOPE: Interactive Ontology Population and Enrichment guided by ontological constraint. Tech. rep. (2021), `https://hal.archives-ouvertes.fr/hal-03177176`
6. Butt, A., Haller, A., Liu, S., Xie, L.: Activeraul: Automatically generated web interfaces for creating rdf data. Semantic Web **2013** (2013)
7. 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)
8. Fang, Y., Cheng, R., Luo, S., Hu, J., Huang, K.: C-Explorer: Browsing communities in large graphs. VLDB (2017)
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. Henry, N., Fekete, J., McGuffin, M.J.: Nodetrix: a hybrid visualization of social networks. TVCG (2007)
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. Springer (2017)
13. 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)
14. Omidvar-Tehrani, B., Amer-Yahia, S.: Data pipelines for user group analytics. In: SIGMOD Conference. pp. 2048–2053. ACM (2019)
15. Rahman, P., Jiang, L., Nandi, A.: Evaluating interactive data systems. VLDB J. **29**(1), 119–146 (2020)
16. Thomas, J.J.: Illuminating the Path: The Research and Development Agenda for Visual Analytics. IEEE Computer Society (2005)
17. Valdestilhas, A., Publio, G., Cimmino Arriaga, A., Riechert, T.: Voceditor an integrated environment to visually edit, validate and versioning rdf vocabularies (2020)
18. Wiens, V., Lohmann, S., Auer, S.: Webvowl editor: Device-independent visual ontology modeling. In: ISWC 2018 Posters & Demonstrations (2018)
19. 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 (2020)