Modélisation et mise à jour d'ontologies interactives : application à la formation par simulation de gestes médicaux

Shadi Baghernezhad Tabasi

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Shadi BAGHERNEZHAD TABASI

Thèse dirigée par **Marie-Christine ROUSSET** et
codirigée par **Fabrice JOUANOT** et **Loïc DRUETTE**

préparée au sein du **Laboratoire Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Interactive Ontology Modeling and Updating: Application to Simulation-based Training in Medicine

Thèse soutenue publiquement le **« date de soutenance »**,
devant le jury composé de :

Mme Danielle ZIEBELIN
Professeur à l'Universite Grenoble Alpes, LIG, Présidente
Mme Nathalie PERNELLE
Professeur à l'Universite Sorbonne Paris Nord, LIPN, Rapporteure
Mr Julien BROISIN
Maître de conférence (HDR) à l'Université Paul Sabatier, IRIT, Rapporteur
Mme Sylvie DESPRÈS
Professeur à l'Universite Sorbonne Paris Nord, LIMICS, Examinatrice
Mme Marie-Christine ROUSSET
Professeur à l'Université Grenoble Alpes, Directrice de thèse
Mr Fabrice JOUANOT
Maître de conférence à l'Université Grenoble Alpes, Co-encadrant de thèse
Mr Loic DRUETTE
Ingénieur d'étude à l'Université Claude Bernard Lyon 1, Co-encadrant de thèse

Résumé

Les ontologies spécialisées servent à capturer le savoir-faire de spécialistes expérimentés dans le domaine concerné, en vue de partager cette expertise au sein d'une plus large communauté dans un but de formation ou d'explications auprès d'utilisateurs moins experimentés. L'objectif principal de cette thèse est la construction d'une ontologie support à la formation par simulation de gestes médicaux, qui est un domaine nouveau encore peu formalisé et pour lequel peu de documentation existe. Ce travail de thèse englobe le cycle de vie complet d'une ontologie: la construction, l'enrichissement, le peuplement, et la mise à jour.

Dans cette thèse, nous décrivons une méthodologie de construction collaborative d'ontologies en 4 étapes qui a débouché sur l'ontologie OntoSAMSEI. La première étape de bootstrap, où un embryon d'ontologie est construite avec l'aide d'un petit nombre d'experts, est suivie par une étape d'élicitation de connaissances auprès d'un plus large panel d'experts à base d'un questionnaire, dont l'analyse des réponses permet de compléter l'ontologie initiale et de la mettre à jour en la peuplant et l'enrichissant. L'ontologie résultante est une hiérarchie de classes et de propriétés, enrichie par un ensemble de contraintes sémantiques sur les classes et les propriétés.

Comme support à la mise à jour d'ontologies, nous avons conçu et implémenté l'environnement IOPE pour la construction automatique d'une interface graphique sous la forme de pages Web pré-remplies. La principale idée sous-jacente à IOPE est de transposer les données et les contraintes de l'ontologie à mettre à jour dans des formulaires pré-remplis en utilisant un ensemble de règles de mappings. Ces pages Web pré-remplies, automatiquement générées à partir de l'ontologie d'entrée, fournissent un guide pour les utilisateurs et facilitent l'exploration et la mise à jour de l'ontologie à l'aide de widgets graphiques interactifs et de règles de bindings permettant de lier les entrées des utilisateurs à des triplets RDF à ajouter à l'ontologie.

Nous avons mené une étude expérimentale poussée auprès d'utilisateurs experts dans le domaine de la formation par la simulation en Médecine pour évaluer l'ontologie OntoSAMSEI, mais aussi l'environnement intéractif IOPE.

Abstract

Specialized ontologies are constructed to capture the skills of experienced experts in a particular domain, with the goal of sharing them with a larger community of trainees and less experienced experts in the domain. The main objective of this thesis is to construct a specialized ontology for the rising domain of simulation-based medical education, where formal models are lacking, and documentations are scarce. The thesis focuses on constructing an accurate and complete specialized ontology, and enriching and populating the constructed ontology.

In this thesis, we have designed a four-staged collaborative ontology engineering methodology, which has resulted in the construction of the ontology called OntoSAMSEI. The first step is ontology bootstrapping (i.e., build a small initial ontology with the help of domain experts), followed by knowledge elicitation (fill the ontology using a questionnaire disseminated among the domain experts), enhancement (improve the core ontology by modeling commonalities), and update (enrichment and population). The resulting ontology is a hierarchy of classes and of properties, enriched by ontological constraints on the properties and on the classes.

As a support to ontology update, we have designed and implemented a framework called IOPE for the automatic construction of a Graphical User Interface (GUI) consisting of pre-filled Web pages. The core idea behind IOPE is to transpose the RDF data and the ontological constraints into a GUI, using a set of mapping rules. These automatically generated GUIs provide guidance for domain experts and facilitate the ontology exploration and update through interactive graphical widgets. To finalize ontology updates, we propose a set of binding rules to specify how to transform user interactions into RDF graphs.

The two contributions of this thesis are evaluated using an extensive and in-depth expert study, to show the benefits of IOPE and OntoSAMSEI in real-world use cases of medical experts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ontologies have been introduced in 1993 by Thomas R. Gruber [Gru93] to formalize the knowledge of a domain in a conceptual model, shareable by domain experts and processable by machines. Nowadays, many ontologies are made available in the Semantic Web [GAVS11], in particular in biology, medicine, and life services, as a result of a collective effort of whole communities.

Ontology construction has been addressed by varied techniques and methodologies, combining text mining and knowledge elicitation from experts [SA11]. Most of the methods focus on well-defined domains such as anatomy, genomics, or medical standard terminologies. However, the current approaches are not well adapted for *ill-defined domains*, i.e., domains in which formal models do not exist and a little standard documentation is available.

Pedagogical domains are examples of such domains, as teaching objectives are hard to formalize, and teaching methods are difficult to share within a common and standardized referential. This is particularly true for the domain of simulation-based medical training which is becoming a central need in medical education. For instance, it is shown in [FBC+17] that simulation-based training in cardiac surgery results in highly efficient and safe training of junior medical students in various modules, such as cardiopulmonary bypass and massive air embolism, to name a few.

So far, only a few pioneering educators have developed and documented a specific expertise for setting up simulation-based training sessions in their speciality. The SAMSEI project[1] (Learning Strategies for Health Professions in Immersive Environment) has been launched by the French Ministry of Higher Education and Research[2] (abbr., MESR) to promote innovative educational programs of excellence. SAMSEI's main objective is to

[1] *http://samsei.univ-lyon1.fr*
[2] *https://www.enseignementsup-recherche.gouv.fr*

set up a simulation-based educational program for participatory and immersive learning [FO15] for students from all health sectors in the University of Claude Bernard Lyon 1[3] and its partners.

This thesis takes place in the context of a collaborative research project between the SAMSEI project and Grenoble Informatics Laboratory (abbr., LIG)[4], aiming to develop methods and tools for simulation-based medical training to be integrated within the SIDES 3.0 platform[5]. The Intelligent Health Education System 3.0 (abbr., SIDES 3.0) is a national e-learning platform based on semantic Web technologies for online medical training in France.

The goal of this thesis is to *develop an interactive and incremental ontology modeling and updating approach for ill-defined domains* such as simulation-based medical education. To achieve this goal, two main challenges have been identified:

1. How can we elicit knowledge for such little formal and documented domains?
2. 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). How can we support the dynamic evolution of an ontology over time?

In Sections 1.1, 1.2, and 1.3, we introduce our strategy and methodology to tackle the above challenges for eliciting and updating the knowledge in ill-defined domains.

1.1 Our Research Focus

In this thesis, we focus on the construction of specialized ontologies that capture the skills of a limited number of experienced experts in a particular domain (such as simulation-based medical education), with the objective of sharing them with a larger community of trainees or less experienced experts in the domain. This engenders the two following research directions:

Direction 1: Knowledge elicitation and formalization of ill-defined domains. In such domains, formal models are lacking and documentations are scarce. Only a few medical trainers have experienced and documented how to set up simulation-based medical training sessions in their medical specialities. In addition, these pioneering

[3] *https://www.univ-lyon1.fr/en*
[4] *https://www.liglab.fr/en*
[5] *https://sides3.uness.fr*

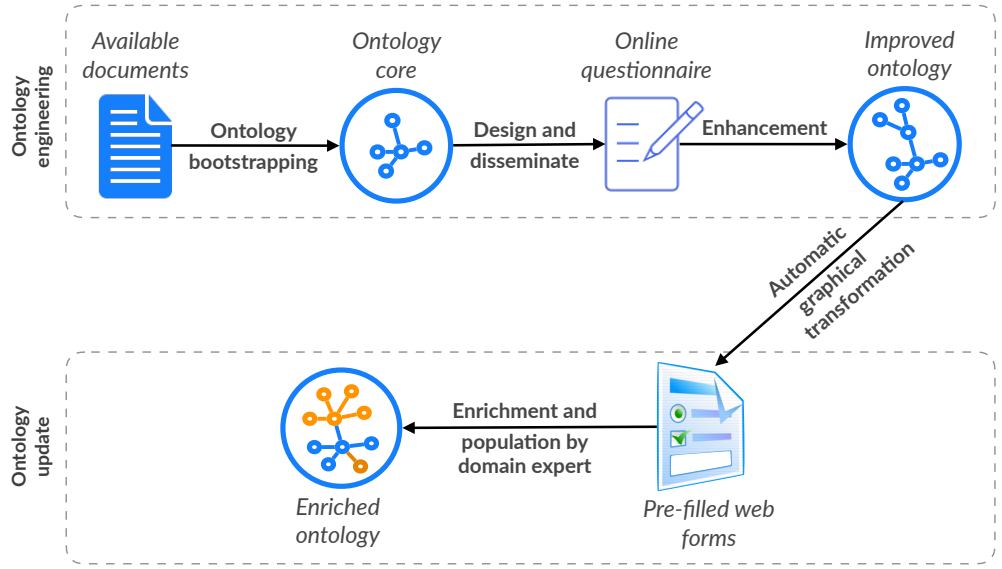

FIGURE 1.1: The five steps of our incremental ontology engineering and update method.

trainers are very busy and not easily accessible. Hence, the initiatives of knowledge elicitation and formalization in these domains are momentous.

Direction 2: Design and development of a tool for interactive ontology update by domain experts. This research direction involves the automatic construction of a Graphical User Interface (GUI) built from a given ontology, as the support of the controlled update process of the input ontology. This task is associated with two main challenges:

1. Ontologies are represented as graphs. The graph structure of ontologies is difficult to graphically present within the user interface consumed by the experts. While there exist several methods to visualize a graph [HFM07, PWC+17, HBZC17, FCL+17], the outcome is often burdensome to digest by domain experts.

2. Moreover, it is challenging to enable the experts to perform ontology updates without requiring to learn the formal syntax and semantics of ontology languages.

1.2 Our Methodology

Following the research directions discussed in the previous section, we contribute an interactive and incremental ontology modeling approach. Figure 1.1 illustrates the proposed methodology, which consists of two principled components: (*i*) incremental ontology engineering and (*ii*) ontology update. It comprises five consecutive steps, mentioned below:

1. **Ontology bootstrapping** is to initialize the core structure of the ontology by employing a few available documents provided by a limited number of experts;
2. **Designing** and **disseminating** an online questionnaire aims to collect more information from several groups of experts;
3. **Enhancement** aims to extend the core ontology using a text mining method on the results of the online questionnaire;
4. **Generation** of a graphical user interface (GUI) made of pre-filled Web forms aims to provide graphical representation of the current ontology;
5. Last, the system **enriches** and **populates** the input ontology by interacting with the graphical representation.

These steps collectively cover the whole pipeline of ontology engineering. In the following, we elaborate on these steps in the context of engineering a specialized ontology in the domain of simulation-based medical education, called OntoSAMSEI [TDJ+21b].

Step 1: Ontology bootstrapping. To construct OntoSAMSEI, first we exploited some reported expertise of a group of pioneer trainers who have documented simulation learning units of various types. As a result, we obtained the core structure of OntoSAMSEI for a few number of simulation-based training sessions, e.g., suture, port implant, blood transfusion, hygiene, etc. The achieved general structure of OntoSAMSEI contains the simplest elements to formalize, e.g., targeted audiences, learning objectives, necessary prerequisites, required resources, risks incurred during the simulation session and in real situation, and evaluation of the prerequisites and objectives.

Step 2: Design and dissemination of an online questionnaire. In order to improve the core ontology, we designed an online questionnaire and disseminated it among the domain experts (i.e., health trainers in simulation learning), and succeeded to acquire accurate descriptions of various simulation sessions. We constructed the online questionnaire using the general structure obtained in the first step of our methodology, with several sections that represent several extracted elements that need to be filled in by the experts. These sections expose the targeted audience, the aimed objectives, the prerequisites, the resources required (human, consumable, simulator, material), risks, as well as the evaluation mode of prerequisites and objectives. Last, we teamed up with a pedagogical engineer to create a directory of 1223 health professionals, and provide face-to-face or remote support for helping them in answering the questionnaire.

Step 3: Enhancement. Next, we improved and refined OntoSAMSEI by grouping the answers by simulation sessions, and extracting the descriptions for each rubric. Then

we employed two simple yet effective text mining approaches, namely "noun phrase extraction" and "part-of-speech (POS) tagging", to find commonalities between the answers. As a result, new sub-classes have been added in the hierarchy of simulation-based training units with their properties and constraints, and existing classes have been enriched with constraints on their properties. The *ontological constraints* are important to consider in the model, because they can be perceived as a guide for setting up future or more specific sessions for each type of simulation-based training units.

Step 4: Automatic graphical transformation of the ontology. In order to facilitate the exploration of an ontology by domain experts (who can be recognized or less experienced experts), we designed a tool named IOPE GUI, that takes as input a set of ontological constraints defining a given ontology, and transforms it into a set of pre-filled Web forms, where the experts can interact with those pages through interactive graphical widgets.

Step 5: Interactive ontology enrichment and population. This last step is supported by the output of the IOPE GUI which enables domain experts to interactively enrich the current ontology by adding new classes and/or constraints, and less experienced experts to be guided for adding new instances of existing simulation-based training units. The input (constraints or instance) entered by domain experts through the IOPE GUI are transformed into RDF triples. These RDF triples must be verified by an ontology engineer before being permanently added in the domain ontology.

Following the five aforementioned steps, we constructed the OntoSAMSEI ontology consisting of the model for simulation-based training sessions. We also provided interactive means (through IOPE GUI) to update the ontology.

1.3 Thesis Contributions

In this thesis, our main contributions can be summarized as follows:

- The construction of a simulation-based medical education domain vocabulary called OntoSAMSEI, by designing an incremental ontology modeling approach. The results of this direction of the thesis have been published in the proceedings of the IEEE International Conference on Semantic Technologies for Smart Information Sharing and Web Collaboration [TDJ+21b], and in the doctoral consortium of the Conference on Knowledge Engineering and Ontology Development (KEOD) [BTRD+19].

- The automatic construction of a Graphical User Interface (GUI) called IOPE, built from the ontological constraints of a given ontology, as the support of the controlled update process of the considered ontology. The results of this direction of thesis has been published in the demo proceedings of the SEMANTiCS conference [TDJ+21a]. A longer version of our work in this direction is also under review.

1.4 Thesis Outline

This thesis is organized into the following chapters. Chapter 2 covers the definition of the relevant notions used in this thesis. Chapter 3 presents the details on the construction of the OntoSAMSEI ontology that we have developed for the simulation-based medical education domain. Chapter 4 describes the interactive ontology enrichment and population approach that we have developed to update the ontology. Chapter 5 presents an extensive set of results on the evaluation of our ontology engineering approach and our ontology update approach. Chapter 6 summarizes the results of our thesis and presents future research directions.

Chapter 2

Preliminaries

In this chapter, we introduce the semantic Web background on which this thesis relies: ontologies in Section 2.1, RDF and SPARQL in Section 2.2, RDFS in Section 2.3, and OWL in Section 2.4. Finally, Section 2.5 summarizes the chapter.

2.1 Ontologies and the Semantic Web

In 1999, Tim Berners-Lee presented for the first time his vision of the Semantic Web: *"The Semantic Web is not a separate Web, but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in co-operation"* [BLHL01]. Berners-Lee's goal was to extend the current Web with metadata by allowing both machines and humans to better manipulate information and make meaningful interpretations. Therefore, we are no longer talking only about a Web of documents but a Web of data and knowledge [BHBL11].

With the emergence of the semantic Web, the notion of ontology has experienced a new rise. Formerly reserved for the field of philosophy, and later for the Artificial Intelligence (AI) domain, ontologies now represent the backbone of the semantic Web technology. In the literature, several definitions of ontology have been presented and evolved over the time. The most quoted is given by Gruber [Gru95]: *"an ontology is an explicit specification of a shared conceptualization"* which is, in turn, *"the objects, concepts, and other entities that are presumed to exist in some domain of interest and the relationships that hold among them which are understandable by humans and processable by machines"*.

Different models of increasing complexity can be distinguished for knowledge conceptualization [Zac07]:

- A *taxonomy* is a classification of terms or concepts of a domain organized in a hierarchical structure.
- A *thesaurus* adds non-hierarchical semantic relationships between terms and concepts such as equivalence relation and association relations, and other properties to each concept or term, such as its preferred or alternative label(s) in SKOS[1]. In addition, concepts and terms can also be organized in collections.
- An *ontology* is a more complex formalization of a domain of interest in which *ontological constraints* are added to refine the semantics of the involved entities and their relationships.
- A *populated ontology* adds the description of instances (or individuals) through class membership assertions and property assertions between instances. The important point is that these assertions together with the ontological constraints can be exploited by inference mechanisms to derive new facts or new knowledge.
- A *knowledge graph*, presented by Google as a new trend in 2012, is a way to represent big populated ontologies [Krö17, EW16].

In terms of coverage, we can distinguish general and specialized ontologies. A general ontology is the upper ontology shared by all domains such as SUMO (Suggested Upper Merged Ontology) [NP01]. Specialized ontologies represent information for specific domains, for which specialized schema must be created to make the data useful in making real world decisions. In many domains, data and/or knowledge are evolving over time. (Populated) ontologies are thus intended to be dynamic structures that are to be updated either periodically or continuously.

Based on the definitions of the semantic Web and ontologies, we can highlight two main benefits of these technologies:

- *Interoperability*, which reposes on sharing and exchanging data across Web applications and agents;
- *Inferencing*, which means the ability of the system to derive new knowledge and new facts.

To make the Semantic Web a reality and lift current Web to its full potential, powerful and expressive representation languages and systems are required. Such web ontology languages must be able to describe and organize knowledge in the Web via a

[1] *https://www.w3.org/TR/skos-reference/*

standard set of special terms in a machine understandable way. On the Web, we currently have three such standards, namely RDF, to represent data and metadata, RDF Schema (RDFS), which defines some basic terms like subClassOf, and the Web Ontology Language (OWL), which defines a much broader range of special terms, including Restrictions, disjointWith, unionOf, and far more. In the following, we describe these semantic Web languages standards.

2.2 RDF and SPARQL

Resource Description Framework (RDF)[2] developed by the World Wide Web Consortium (W3C) is a graph data model for representing and structuring data and metadata. It is based on the idea of making statements about resources (in particular web resources). Each RDF statement is a three-part structure (subject, predicate, object), known as triples. For example, one way to represent the notion "Coronavirus disease 2019 (covid-19) has dry cough symptom" in RDF is the following triple: `(covid-19 hasSymptom dry_cough)` a subject denoting "the covid-19", a predicate denoting "has symptom", and an object denoting "dry cough".

The subject in a RDF triple represents the resource to be described and it is either an Internationalized Resource Identifier (IRI) or a blank node. The predicate is an IRI which also indicates a resource (property), representing a relation between subject resources and object resources. The object is an IRI of some other resource, blank node or a literal value. IRIs are standard identifiers used for denoting any Web resource involved in RDF statements. The IRIs in an RDF vocabulary often begin with a common substring known as a namespace IRI. Some namespace IRIs are associated by convention with a short name known as a namespace prefix. In some serialization formats it is common to abbreviate IRIs that start with namespace IRIs by using a namespace prefix in order to assist readability. A *literal* is a string that represents a specific value such as strings, numbers, and dates 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`.

Definition 2.1 (RDF graph)**.** Let I, L and B be countably infinite pairwise disjoint sets representing respectively *IRIs*, *literals* and *blank nodes*. An *RDF graph* is a finite set of *RDF triples* (s, p, o), where $(s, p, o) \in (I \cup B) \times I \times (I \cup L \cup B)$.

RDF statements are composed of two high-level types of conceptual elements: *properties* and *classes*. Properties are the relationships that hold between pairs of resources. Classes are groups of resources of the same type with some conceptual similarities. A member

[2] *Resource Description Framework (RDF): https://www.w3.org/RDF/*

of a class is often called an *instance* of that class. The relationship between instances and classes in RDF is defined using the property `rdf:type`.

RDF statements can be represented in a variety of syntax notations and data serialization formats, among which RDF/XML[3], Terse RDF Triple Language (Turtle)[4], and N-Triples[5].

SPARQL Protocol and RDF Query Language (SPARQL) is the standard query language and protocol to retrieve and manipulate RDF graphs.[6] SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. Each element of a triple pattern (the subject, predicate and object) can be a variable. SPARQL describes, in the query, variables of a pattern to match and their values to use for filtering, and extracts the sub-graphs that match the given pattern from an entire RDF graph. The matching process results in the corresponding values of the specified variables. The SPARQL language specifies four different query variations for different purposes.

- SELECT: Returns all, or a subset of, the variables bound in a query pattern match.
- ASK: Returns true if a query pattern matches, false otherwise.
- CONSTRUCT: Returns an RDF graph constructed by substituting variables in a set of triple templates.
- DESCRIBE: Returns an RDF graph that describes the resources found.

The following is an example of a SPARQL query (SELECT variant) to answer the following question: What are the top 10 countries in Europe with the most death per case ratio for the COVID-19 pandemic?

[3] *https://www.w3.org/TR/rdf-syntax-grammar/*
[4] *https://www.w3.org/TR/turtle/*
[5] *https://www.w3.org/TR/n-triples/*
[6] *SPARQL Protocol and RDF Query Language: https://www.w3.org/TR/rdf-sparql-query/*

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  covid19: <http://www.example.fr/covid19#>
SELECT  ?country
WHERE
{
    ?country rdf:type :EuropeanCountries .
    ?country covid19:confirmedCase ?x .
    ?country covid19:death ?y .
    BIND (100*(?y / ?x) AS ?fatalityRate)
}
ORDER BY DESC(?fatalityRate)
LIMIT 10
```

Two main standards have been proposed for defining semantics over RDF data: RDF Schema (RDFS) and the Web Ontology Language (OWL). While RDFS suffices to express the semantic definitions needed to automate the deductions, more complex definitions are supported by the OWL standard. The following sections first deal with RDFS then with OWL.

2.3 Resource Description Framework Schema (RDFS)

Resource Description Framework Schema (RDFS)[7] is a vocabulary description language to extends the RDF vocabulary with a set of novel terms that form the RDFS vocabulary. RDFS describes properties and classes of RDF resources, with a semantics for generalization hierarchies of such properties and classes. Classes are identified by IRIs and are described using the RDF Schema resources `rdfs:Class` and `rdfs:Resource`. An RDF schema (RDFS) provides (*i*) abstraction mechanisms, such as (multiple) class subsumption `rdfs:subClassOf` or property subsumption `rdfs:subPropertyOf` and (multiple) classification of resources; (*ii*) `rdfs:domain` and `rdfs:range` class specifications to which properties can apply; (*iii*) documentation facilities for names defined in a schema such as `rdfs:label` property that relates a resource with a human-readable label giving its name, etc.

RDFS allows for specifying the following ontological constraints on classes and properties:

- **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`,

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

i.e., that every instance `i` of `C` is an instance of `D`:
$\forall$ `i` (`(i rdf:type C)` $\Rightarrow$ `(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., any pair of resources related by `p` must also be related by `q`:
$\forall$ `i` $\forall$ `j` (`(i p j)` $\Rightarrow$ `(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.:
$\forall$ `i` $\forall$ `j` (`(i p j )` $\Rightarrow$ `(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.:
$\forall$ `i` $\forall$ `j` (`(i p j)` $\Rightarrow$ `(j rdf:type D)`)

As an example, the `hasSymptom` property is sub-property of the `hasMedicalIssue` property and it has `Disease` and `Symptom` classes as domain and range of this property. The `hasMedicalIssue` property has another sub-property `hasSign`. The `Symptom` class have also three sub-classes such as `ChronicSymptom`, `RelapsingSymptom`, and `RemittingSymptom` [Kin68]. Chronic symptoms tend to recur over a long period of time. Remitting symptoms are ones that improve or disappear, and relapsing symptoms are ones that were considered to be resolved but have returned.

2.4 Web Ontology Language

The Web Ontology Language (OWL)[8] is a family of knowledge representation languages for publishing and sharing ontologies on the World Wide Web. OWL extends RDFS with richer ontological constraints. "OWL 2"[9] is an extension and revision of the first version of OWL (referred as "OWL 1") developed by the W3C Web Ontology Working Group, published in 2004.

Features of OWL are a collection of expressive operators for concept description including intersection, union and complement operators, plus explicit quantifiers for properties and relationships, and the ability to specify characteristics of properties, such as transitivity or functionality, etc.

[8] *Web Ontology Language (OWL): https://www.w3.org/OWL/*
[9] *Web Ontology Language (OWL 2): https://www.w3.org/TR/owl2-overview/*

OWL ontological constraints: A main feature of OWL is to define new classes from existing ones by defining restrictions on the members that it may contain. It includes restrictions on the value that is taken for a given property, on the class to which a value belongs on a given property, and on the number of values taken for a given property. Such features enable increasingly complex class definitions.

The language construct in OWL for creating new class descriptions based on descriptions of the prospective members of a class (instances) is called the *property restriction*. OWL distinguishes two kinds of property restrictions: *value constraints* and *cardinality constraints*. Property restrictions have the general form shown as follows:

```
<owl:Restriction>
    <owl:onProperty rdf:resource="(some property)" />
    (precisely one value or cardinality constraint)
</owl:Restriction>
```

OWL has its own class construct, `owl:Class`. The class `owl:Restriction` is defined as a subclass of `owl:Class`. The `owl:Restriction` class is used in association with a blank node class, and some specific restriction properties used for defining the new class. The restriction class should also have exactly one triple that represents the value constraint or cardinality constraint on the property under consideration. Property restrictions can be applied both to datatype properties (properties for which the value is a literal) and object properties (properties for which the value is an individual).

2.4.1 Value Constraints

OWL provides four value constraint properties which are `owl:someValuesFrom`, `owl:allValuesFrom`, `owl:hasValue`, and `owl:oneOf`. Each describes how the new class is constrained by the possible asserted values of properties. In the following, we will describe their semantics and syntaxes in RDF triples and/or RDF graphs.

OWL:someValuesFrom : The value constraint `owl:someValuesFrom` is a built-in OWL property that links a restriction class to a class description or a data range. A restriction containing an `owl:someValuesFrom` constraint is used to describe a class C of all individuals for which at least one value of the property P is an instance of the class description D or a data value in the data range. The semantic of the restriction `(C: P someValuesFrom D)` is as follows:

$$\forall \texttt{i}(\texttt{(i rdf:type C)} \Rightarrow \exists \texttt{j}(\texttt{(j rdf:type D)} \wedge \texttt{(i P j)}))$$

This restriction can be written in RDF using several RDF triples shown as follows:

```
(C rdfs:subClassOf _:b1)
(_:b1 rdf:type owl:Restriction)
(_:b1 owl:onProperty P)
(_:b1 owl:someValuesFrom D)
```

As an example, we could define a class for Coronavirus disease `(CoronavirusDisease)`, that has some values from the class `RemmitingSymptom` on the property `hasSymptom`. Figure 2.1 shows the RDF graph representation of this restriction in the form of `(CoronavirusDisease: hasSymptom someValuesFrom RemmitingSymptom)`.

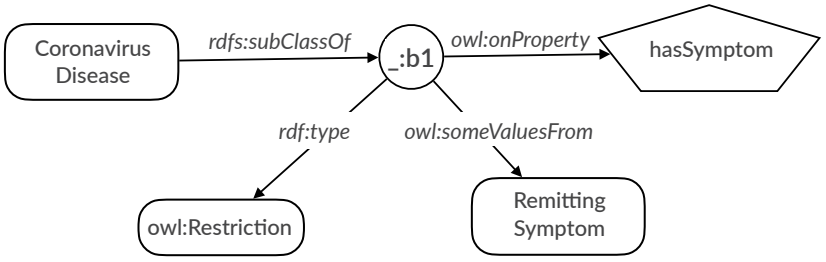

FIGURE 2.1: The `owl:someValuesFrom` RDF graph for `CoronavirusDisease` class on the property `hasSymptom`.

OWL:allValuesFrom : The value constraint `owl:allValuesFrom` links a restriction class to either a class description or a data range. A restriction containing an `owl:allValuesFrom` constraint is used to describe a class C of all individuals for which all values of the property P are either instances of the class description D or are data values within the specified data range.

The semantic of the restriction `(C: P allValuesFrom D)` is as follows:

$$\forall i \forall j((i \text{ rdf:type } C) \wedge (i \ P \ j) \Rightarrow (j \text{ rdf:type } D))$$

This restriction can be written in RDF using several RDF triples shown as follows:

```
(C rdfs:subClassOf _:b1)
(_:b1 rdf:type owl:Restriction)
(_:b1 owl:onProperty P)
(_:b1 owl:allValuesFrom D)
```

OWL:hasValue : The third kind of restriction in OWL is called `owl:hasValue`. As in the other two restrictions, it operates on a particular property as specified by `owl:onProperty`. This value constraint links a restriction class to a value v, which can be either an individual or a data value. `owl:hasValue` is leveraged to produce a restriction class C whose description is of the form "All individuals for which the

property P has at least one value semantically equal to v (it may have other values as well)" and we will denote it by `(C: p value v)` throughout the thesis. The semantic of this restriction is as follows:

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

This restriction can be written in RDF using several RDF triples shown as follows:

```
(C rdfs:subClassOf _:b1)
(_:b1 rdf:type owl:Restriction)
(_:b1 owl:onProperty P)
(_:b1 owl:hasValue v)
```

For example, as shown in Figure 2.2, the class `CoronavirusDisease` has the value `respiratory_droplets` on the property `transmittedBy`. The `respiratory_droplets` is an instance of `AerosolTransmission` class.

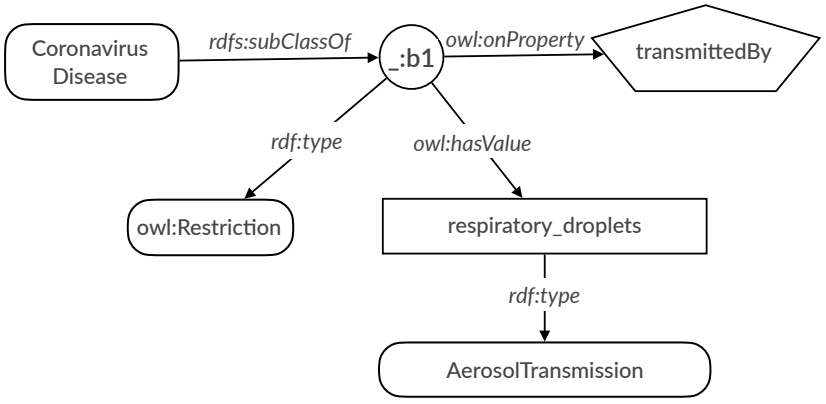

FIGURE 2.2: The `owl:hasValue` RDF graph for `CoronavirusDisease` class on the property `transmittedBy`.

Formally, the `hasValue` restriction is just a special case of the `someValuesFrom` restriction, in which the class D is a singleton ($D = \{v\}$). Although it is just a special case, `owl:hasValue` has been identified in the OWL standard in its own right, because it is a very common and useful modeling form.

OWL:oneOf : OWL provides the means to specify a class via a direct enumeration of its members with the `owl:oneOf` property. The value of this built-in OWL property must be a list of individuals which collectively form the instances of a class. The class extension of a class described with `owl:oneOf` contains exactly the enumerated individuals. The semantics of the restriction `(C: p oneOf {v1,...,vn})` is as follows:

$$\forall \texttt{i}(\texttt{(i rdf:type C)} \Rightarrow \bigvee_{k \in [1..n]} \texttt{(i p vk)})$$

As an example, we could define the class aerosol transmission (`AerosolTransmission`) as equivalent to the class with the following members: {`respiratory_droplets`, `airborne_droplets`}. The following syntax is the set of RDF triples representing the (`AerosolTransmission`) class.

```
AerosolTransmission owl:equivalentClass
[
    owl:oneOf ( respiratory_droplets  airborne_droplets )
]
```

The combination of `owl:oneOf` and `owl:someValuesFrom` provides a generalization of `owl:hasValue`. Whereas `owl:hasValue` specifies a single value that a property can take, `owl:someValuesFrom` combined with `owl:oneOf` specifies a distinct set of values that a property can take.

2.4.2 Cardinality Constraints

Up till now, we have seen restrictions that define classes based on the presence of certain values for given properties. OWL allows another type of ontological constraints, based on the number of values a property can take. Such a restriction is called a "cardinality restriction". In OWL, like in RDF, it is assumed that any instance of a class may have an arbitrary number (zero or more) of values for a particular property. To make a property required (at least one), to allow only a specific number of values for that property, or to insist that a property must not occur, cardinality constraints can be used. All cardinality constraints can be qualified or unqualified, i.e., in the former case, the cardinality constraint only applies to individuals that are connected by the property and are instances of the qualifying class, and in the latter case, the restriction applies to all individuals that are connected by the property (this is equivalent to the qualified case with the qualifying class equal to `owl:Thing`). OWL provides three constructs `MinCardinality`, `MaxCardinality`, `ExactCardinality`, which constrain those individuals that are connected by a property to at least, at most, and exactly a given number of values (individuals or data values) of a specified class expression, respectively.

The semantics of these restrictions are as follows:

- **`MinCardinality`** denoted by `(C: p min k D)`

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

- **MaxCardinality** denoted by `(C: p max k D)`

$$\forall \mathtt{i} \forall o_1, \ldots, o_{k+1} (\mathtt{(i\ rdf:type\ C)} \wedge \bigwedge_{j \in [1,\ldots,k+1]} \mathtt{(i\ p}\ o_j\mathtt{)} \wedge \mathtt{(}o_j\ \mathtt{rdf:type\ D)}) \Rightarrow \bigvee_{j \neq l, j, l \in [1,\ldots,k+1]} (o_j = o_l)$$

- **ExactCardinality** denoted by `(C: p exactly k D)`. Exact cardinality is the intersection of min cardinality and max cardinality.

The syntax for three different cardinality constraint constructs is similar. Here are the RDF triples for qualified restrictions:

```
(C rdfs:subClassOf _:b1)
(_:b1 rdf:type owl:Restriction)
(_:b1 owl:minQualifiedCardinality/owl:maxQualifiedCardinality/owl:
    qualifiedCardinality  "k"^^xsd:nonNegativeInteger)
(_:b1 owl:onProperty P)
(_:b1 owl:onClass D)
```

Here are the RDF triples for unqualified restrictions:

```
(C rdfs:subClassOf _:b1)
(_:b1 rdf:type owl:Restriction)
(_:b1 owl:minCardinality/owl:maxCardinality/owl:Cardinality  "k"^^xsd:
    nonNegativeInteger)
(_:b1 owl:onProperty P)
```

As an example shown in Figure 2.3, the class `CoronavirusDisease` has at least one value for the property `diagnosedWith` on the `ViralTesting` class.

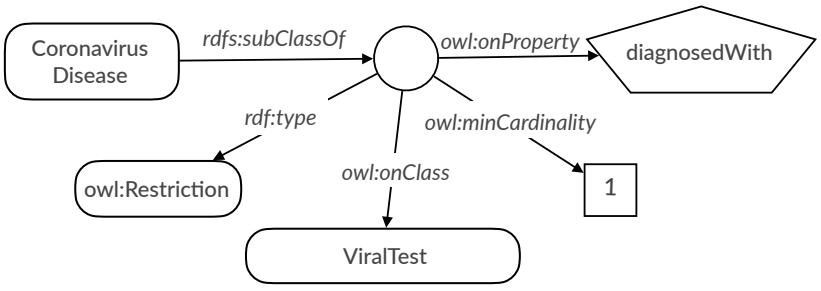

FIGURE 2.3: The `owl:minQualifiedCardinality` RDF graph for `CoronavirusDisease` class on the property `iagnosedWith` and the class `ViralTesting`.

TABLE 2.1: RDFS and OWL constraints considered in this thesis.

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

OWL provides the facility to use any natural number as a cardinality. The particular restrictions of cardinalities to the numbers 0 and 1 have special modeling utility as follows [AH11]:

- **minCardinality 1:** The restriction of the `minCardinality` to 1 indicates the set of individuals for which some for the specified property is required. The Restriction `onProperty consumeTreatment minCardinality 1` explicitly specifies the set of individuals that consume at least one treatment.
- **maxCardinality 1:** The restriction of `maxCardinalilty` to 1 specifies that a value is unique (but need not exist). The restriction `onProperty consumeTreatment maxCardinality 1` explicitly specifies the set of individuals who consume at most one treatment. In other words, they have limited themselves to a single treatment.
- **minCardinality 0:** The restriction of the `minCardinality` to 0 describes a set of individuals for which the presence of a value for the `onProperty` is **optional**. In the semantics of OWL, this is superfluous (since properties are always optional anyway), but the explicit assertion that something is optional can be useful for model readability. The restriction `onProperty consumeTreatment minCardinality 0` explicitly specifies the set of individuals for which consuming a treatment is optional.
- **maxCardinality 0:** The restriction of the `maxCardinality` to 0 indicates the set of individuals for which no value for the specified property is allowed. The Restriction `onProperty consumeTreatment maxCardinality 0` explicitly specifies the set of individuals that consume no treatment.

2.5 Summary

In this chapter, we provided some background view on the notion of semantic Web, semantic Web languages and ontological constraints. All these notions are the backbone of this thesis, since we intend to propose an approach to interactively model and update ontologies guided by ontological constraints. Table 2.1 summarizes all the ontological constraints (RDFS and OWL constraints) that we consider in this thesis.

Chapter 3

Ontology Engineering for Simulation-based Training in Medicine

3.1 Introduction

The domain of Medicine has traditionally relied on an apprentice-style approach to learning and experience [Lat10]. This inevitably exposes patients to inexperienced healthcare practitioners, and the dangers and harm associated with this practice are increasingly unacceptable [Lat10]. Simulation "is a technique to replace or amplify real-patient experiences with supervised and guided experiences, artificially contrived to evoke or replicate substantial aspects of the real world in a fully interactive manner" [Gab04]. Simulation-based medical education enables knowledge, skills, and attitudes to be acquired for all healthcare professionals in a safe, educationally orientated, and efficient manner. Simulation-based training initially began with life-like manikins and now encompasses an entire range of systems, from synthetic models all the way to high fidelity[1] simulation suites.

Although the simulation-based training is required as a critical part of healthcare education and training [AMD+10], many healthcare educators are uncertain how best to integrate simulation training into their programs and curricula. So far, only a few pioneering healthcare educators have developed and documented a scarce pedagogical expertise for setting up some training sessions based on simulation. We note that in all medical fields, loss in accuracy is not expected and tolerated. Ontology modeling for

[1] *In Healthcare Simulation, the manikins that most closely resemble human anatomy are designated as high-fidelity manikins*

simulation-based training domain is indeed an accurate way to design and share simulation teaching units with less experienced educators and trainees, but at the same time raises difficult issues of knowledge acquisition.

Despite the plethora of available Information Extraction (IE) tools and ontology learning techniques which facilitate the knowledge acquisition and the transition from unstructured data to organized knowledge [ea20, NGJ+19], a majority of these approaches are based on many underlying assumptions, mentioned below:

- First, it is assumed that the raw text is available, i.e., the task of obtaining such text from diverse sources (Web pages, text documents, PDF documents, etc.) is ensured;
- It is also assumed that certain loss in accuracy is both expected and tolerated;
- Some agreement is also assumed to exist, i.e., there is a universal truth about what constitutes correct knowledge;
- Last, it is also assumed that the examples are readily available or easily obtainable to train the models.

These assumptions are not satisfied in the domain of simulation-based training in Medicine where expertise is in the hands of few pioneering teachers who have experienced the setting up of simulation-based training sessions in some specific medical specialities.

In this chapter, we focus on the construction of specialized ontologies that capture the skills of experienced experts in a particular domain, with the objective of sharing them with a larger community of trainees or less experienced experts in the domain. This is particularly the case for domains related to pedagogy, because teaching objectives are hard to formalize and teaching methods are also hard to share within a common and standardized referential.

This chapter is organized as follows: In Section 3.2, we present the related work on ontology engineering. Then in Section 3.3, we detail our approach for the construction of an ontology in the simulation-based medical education domain. We summarize the chapter in Section 3.4.

3.2 Related Work

Ontology engineering (OE) studies the "activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for

building ontologies" [GPFLC06]. As Guarino and Oberle emphasize the collaboration among stakeholders towards engineering commonly agreed ontologies [GOS09], Kotis et al. [KVS20] assert that, ideally, an ontology engineering methodology (OEM) must support all involved stakeholders during the ontology life cycle, i.e., from the development of seed ontologies to the continuous evolution and maintenance, keeping them "live" to shape knowledge. OEMs can focus on developing ontologies from scratch or offer support in the identification of ontological (and non-ontological) resources to be reused, or can highlight some particular aspects of the development process.

Since 1996, several surveys on ontology engineering methodology have been written [UG$^+$96, FLGP02, CFLGP03, IMM$^+$13, SSD13, SLR14, YNDJ16, KVS20] in order to represent different aspects of a specific domain. Methodologies differ widely in their development approach, stakeholders' participation in the development process, details in defining the tasks, and the support provided in various stages. According to [KVS20], OEMs can be divided in three broad categories, depending on the types of actors involved in the ontology engineering process:

- **Non-collaborative OEMs:** This category encompasses those OEMs that do not emphasize the cooperation among stakeholders towards engineering commonly agreed ontologies. These methodologies provide, in a systematic and formal way, the set of phases, tasks, and workflows which are necessary to develop an ontology. Representative examples are METHONTOLOGY [FLGPJ97], Uschold and King's methodology [UK95], On-To-Knowledge Methodology (OTKM) [SSSS01] and FMCLGO [GDMF12].

- **Collaborative OEMs:** The methodologies in this category define, also in a systematic and formal way, the set of steps (phases, tasks, and workflows) necessary to develop the ontology. Moreover, the methodologies emphasize the active involvement of domain experts as well as knowledge engineers in all phases of OE (specification, implementation, exploitation, and evaluation). A continuous cooperation toward a commonly agreed knowledge (and its formalization) is pivotal for the success of these OEMs. A collaborative OEM generally comprises three main phases: (*i*) ontology specification, (*ii*) ontology development and (*iii*) ontology exploitation and evaluation phase. Representative examples of such methods are Holsapple and Joshi's methodology [HJ02], HCOME [KV06], DILIGENT [VPTS05], DOGMA-MESS [DMDLM06], UPON-Lite [DNM16], and NeOn [GPSF09]. These methodologies will be introduced later in this section.

- **Custom OEM:** The methodologies in this category do not necessarily define phases, tasks and workflows in a formal and systematic way. However, they still

engage the active involvement of communities of practice and the use of tools (collaborative or not), such as Wiki and GitHub technologies, towards developing ontologies in an agile, decentralized, and most of the times collaborative manner. Representative approaches are included in YAGO [RSH+16], MedRed [CDH+17], Salatino et al. [STM+18], and Arndt et al. [ANR+19].

Our focus in this thesis is on *ill-defined domains*, i.e., domains where formal models are lacking, and documentations are scarce. Collaborative OEMs are the greatest fit for such domains, as they involve different actors to compromise the scarcity of resources. The team developing the shared ontology via a collaborative methodology, consists of stakeholders with different (and probably divergent) interests and complementary competencies. The roles involved in the development team are typically the followings [GPFLC06]:

- **Domain experts.** These members have the knowledge/expertise of the domain and/or data sources. Oftentimes, they are practitioners who are not acquainted with ontology languages, specifications, etc.
- **Ontology engineers.** These members have the knowledge/expertise to design and build ontological specifications and coordinate an OE task.
- **Pedagogical engineers.** The members act as a bridge between the domain experts and ontology engineers, by ensuring a complete transition of knowledge from the former group to the latter.
- **Knowledge engineers.** These members exploit the ontology in "operational" conditions, to solve problems, or perform data-driven analysis tasks. Knowledge engineers can be domain experts, but this is not necessarily the case.

In the following, we describe some recent collaborative OEMs which satisfy our basic requirements for the continuous evolution during their life cycle.

Holsapple and Joshi's OEM. In [HJ02], Holsapple and Joshi propose the first comprehensive methodology for collaborative ontology design based on a Delphi-like approach [LT+75] to structure the consensus-building process. The main objective of this work is to support the creation of a static ontology. First, an initial ontology is developed by merging or integrating existing ontologies. Then the ontology is extended and modified based on the feedback from a panel of domain experts. The engineering process is divided into four phases: (*i*) preparation by defining design criteria and boundaries; (*ii*) first ontology production for participants after orientation; (*iii*) iterative improvement of the first ontology via feedbacks collected from experts; (*iv*) application of the ontology in a specific domain.

NeOn. In [SF10, GPSF09], a scenario-based methodology is proposed to support the collaborative aspects of ontology construction. NeOn emphasizes the development of ontology networks as well as the reuse of existing ontological and non-ontological resources to the development of an ontology [KVS20]. This methodology is based on the analysis of a set of nine ontology development scenarios: (*i*) from specification to implementation, encompassing all the core activities related to engineering; (*ii*) reusing and re-engineering non-ontological resources; (*iii*) reusing ontological resources; (*iv*) reusing and re-engineering ontological resources; (*v*) reusing and merging ontological resources: ontology matching tools enable ontology aligning or merging; (*vi*) reusing, merging and re-engineering ontological resources; (*vii*) reusing ontology design patterns (ODPs); (*viii*) restructuring ontological resources; and (*ix*) localizing ontological resources to translate all the terms of the ontology into another natural language.

HCOME. In [KV06, KP10], Human-Centered Ontology Engineering Methodology (HCOME) is proposed as a human-centred approach for the collaborative engineering of ontologies, where the active participation of knowledge engineers in the ontology life cycle, in close collaboration to domain experts and ontology engineers, is emphasized. An iterative approach to the execution of tasks at all three phases (specification, conceptualization, exploitation/evaluation) is supported, such as the discussion of conceptualizations, and detailed versioning of evolving specifications. A data-driven and bottom-up conceptualization approach is also proposed in the updated version of the HCOME methodology [KP10], supported by learning seed ontologies (knowledge in that case is extracted from query logs).

DILIGENT. Akin to HCOME, DILIGENT method (DIstributed, Loosely-controlled and evolving Engineering of oNTologies) [VPTS05] focuses on a user-centric ontology development which is divided into several phases to be carried out in multiple iterations: (*i*) build collaboratively the first version of an ontology (seed ontology) through a core team of domain experts, users, knowledge engineers, and ontology engineers; (*ii*) adapt the ontology locally to the specific requirements of each knowledge engineer in his/her respective environment; (*iii*) analyze local branches of the shared ontology with respect to their mutual differences via an ontology engineering board; (*iv*) introduce a new version of the shared ontology based on changes agreed in the previous phase; (*v*) update local ontologies locally by knowledge engineers via reusing new terms instead of using their previously defined local terms. As it is obvious, the approach stresses on the distributed and collaborative construction of ontologies.

DOGMA-MESS. The work in [DMDLM06, DLD08] is an extension of the Dogma methodology, called DOGMA-MESS, toward inter-organizational support. It is a collaborative OEM that supports the modeling of shared ontologies in stakeholders' own

terminology and context. To accomplish that, four modes of knowledge conversion are introduced: socialization, externalization, combination, and internalization. Technically, at the core of the approach sits an Ontology Server, which is embedded in a central ontology evolution support system. There are three types of participants: the core domain expert, the domain expert, and the ontology engineer. The ontology evolution process is driven by social knowledge conversion modes. This process is iteratively performed until an optimum trade-off between differences and commonalities of organizational and common perspectives is reached.

UPON-Lite. The OEM proposed in [DNM16] fosters the active role of stakeholders in knowledge identification and definition, through a participative social approach supported by easy-to-use tools (e.g., diagrams, spreadsheets, etc). The role of the domain experts is primary in gathering information and defining and conceptualizing the domain(s), while the role of ontology engineers is sidelined, as they interfere almost at the end of the process to produce the formal model. UPON-Lite is composed of six steps, each of which produces an output that is enriched and refined in the consecutive step: (*i*) *domain terminology* (identification of the main terms of the domains under investigation), (*ii*) *domain glossary* (the definitions of the terms), (*iii*) *taxonomy* (the hierarchical organization) (*iv*) *predication* (connect terms representing properties to the entities they characterize), (*v*) *parthood (meronymy)* (identification of complex entities and their components), and (*vi*) *ontology* (generation of the ontology in a formal language). The process of ontology building and management is carried out on a social media platform. Provided examples are based on Google Docs suite[2], in particular with shared Google Sheets for OE, in conjunction with Google Forms and Google+ for other functions, such as debating and voting. This methodology follows a data-driven and bottom-up approach to OE.

As discussed in [CFLGP03, KVS20], the aforementioned methodologies focus on different aspects of ontology engineering, mentioned below:

- **Ontology creation/reuse.** Some methodologies are designed to build ontologies from scratch (as in DILIGENT and UPON-Lite), or promote the reuse/merge of existing ontologies, (as in Holsapple and Joshi's OEM, NeOn, HCOME, and DOGMA-MESS). The process of ontology building and management in UPON-Lite is carried out on a social media platform, while DILIGENT proposes to generate the seed ontology from the result of a quick agreement between all participants on the high-level terms.

[2] *https://www.google.com/docs/about/*

- **Degree of application dependency.** DOGMA-MESS is application-dependent, because the ontology is built based on a given application. Holsapple and Joshi's OEM and HCOME are instances of semi-dependent methods. DILIGENT is an application-independent method, as its ontology development process is totally independent of the utilisation of the ontology.
- **Strategies for identifying concepts.** Holsapple and Joshi's OEM is a top-down approach. HCOME, DOGMA-MESS, and UPON-Lite are instances of a bottom-up approach. Also DILIGENT is a middle-out approach.

It is argued in [ABGRA14] that the choice of an effective methodology for building ontologies is a difficult task. On one hand, proposed OEMs are not unified, and each group of experts in different entities applies its own approach [CFLGP03]. On the other hand, real-world scenarios require customizable methods, while the majority of the methods propose a pre-defined workflow [ST06]. Thus, it is important to understand the features guiding the choices of OEM (and its management) which are aligned with the requirements elicited with the stakeholders, in terms of feasibility, roles and expertise, and possible scenarios of application and reusability.

3.3 Our Ontology Design Approach

By acknowledging the unavailability of domain-related documentations and requirements, the inaccessibility to the community of contributors, the gradual elicitation of the domain knowledge, the required level of rigor in quality control, and the complexity of the representation, we employ a mixture of DILIGENT and UPON-Lite as our engineering methodology for the development of the simulation-based medical ontology. We adapted these methodologies to build OntoSAMSEI ontology based on the participation of several geographically dispersed experts, with different and complementary skills, in all steps of ontology engineering. Our contribution consists of a methodology with four steps, illustrated in Figure 3.1.

In the following, we describe each step of the OntoSAMSEI process.

3.3.1 Ontology Bootstrapping

The process begins by core domain experts and ontology engineers, where they build an initial ontology with the outputs elicited from the stakeholders meeting and reported expertise of a group of pioneer trainers (domain experts). Those domain experts have

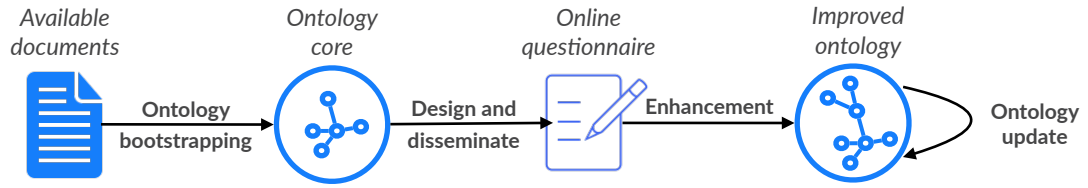

FIGURE 3.1: The four steps of our incremental ontology engineering method

documented simulation learning units of various types. The team involved in building the initial ontology was intentionally relatively small, in order to find a manageable and consensual first version of the ontology, more easily and quickly.

The outcome of the first step is the core structure of OntoSAMSEI for a few number of simulation learning sessions, i.e., suture, port-a-cath placement, blood transfusion, and hygiene. We ratify the simplest elements to formalize, e.g., targeted audiences, learning objectives, necessary prerequisites, required resources (humans, materials, consumables, and simulators), risks incurred during the simulation session and in real situation, and evaluation of the prerequisites and objectives. The obtained general structure of OntoSAMSEI contains 111 classes, 34 properties, and 121 instances. Figure 3.2 illustrates a part of the hierarchies of properties and classes declared in the core OntoSAMSEI, where yellow circles denote the classes, blue boxes denote object properties, and green boxes denote data type properties.

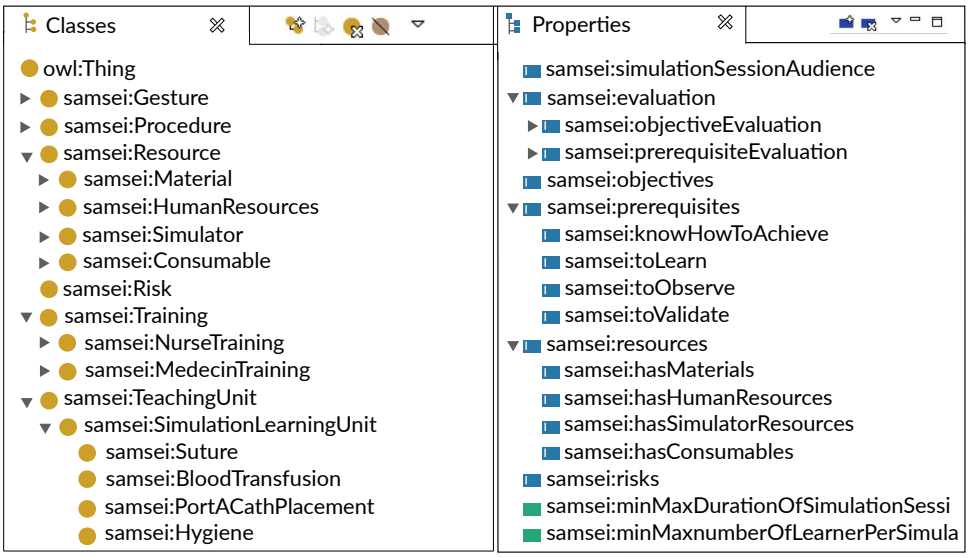

FIGURE 3.2: A part of the hierarchy of properties and classes in the core OntoSAMSEI ontology, visualized in TopBraid Composer.

Several properties assign `samsei:SimulationLearningUnit` class as their domain. The property `samsei:simulationSessionAudience` defines the targeted audience of the simulation learning units, and links the above class to the `samsei:Training`

class. The property `samsei:objectives` defines the goals and objectives of the learning units, and links those units to two classes of `samsei:Procedure` and `samsei:Gestures`. The objectives define what need to be learned by performing the actions, i.e., procedures and gestures. The procedures include different actions in the training, and the gestures define the physical steps to follow in each procedure. The property `samsei:prerequisites` defines what needs to be done before the learning unit begins. It contains four sub-properties: `samsei:knowHowToAchive` (what knowledge needs to be achieved), `samsei:toLearn` (what procedure(s) and/or gesture(s) need to be learned), `samsei:toObserve` (what material(s) should be observed, e.g., watching a pedagogical video), and `samsei:toValidate` (what other simulation learning units should be validated). The property `samsei:evaluation` defines the different types of assessments for the learning units, including the assessment of their objectives and prerequisites. The property `samsei:resource` defines various resources needed for the learning unit, including `samsei:Material`, `samsei:HumanResource`, `samsei:Simulator`, and `samsei:Consumable`.

3.3.2 Knowledge Acquisition by an Online Questionnaire

In contrast to known ontology engineering methodologies available in the literature [UK95, CDH$^+$17, HJ02], our focus is on the ontology development for ill-defined domains. Hence, we require online ontology engineering support to identify and elicit domain knowledge. Therefore, we teamed up with a pedagogical engineer to design an online questionnaire and disseminate it among health educators in the domain of simulation learning to acquire the most accurate description of various simulation learning units.

We have designed the online questionnaire, based on the (generic) properties declared for the specific training sessions described in the bootstrap step, with as many sections in the questionnaire as properties to be filled for a training session. These sections expose respectively the targeted audience, the aimed objectives, the prerequisites, the resources required (human, consumable, simulator, material), the evaluation mode of prerequisites and objectives, as well as the associated risks. There is also a section to ask experts whether they are willing to continue their collaboration with our project. Figure 3.3 illustrates the part of the questionnaire corresponding to the required resources. The full questionnaire is presented in Appendix A.

For the campaign of the questionnaire dissemination, we created a directory listing 1223 health professionals, and provided face-to-face or remote support to help them in answering the questionnaire. In the course of 7 months, we received 304 responses,

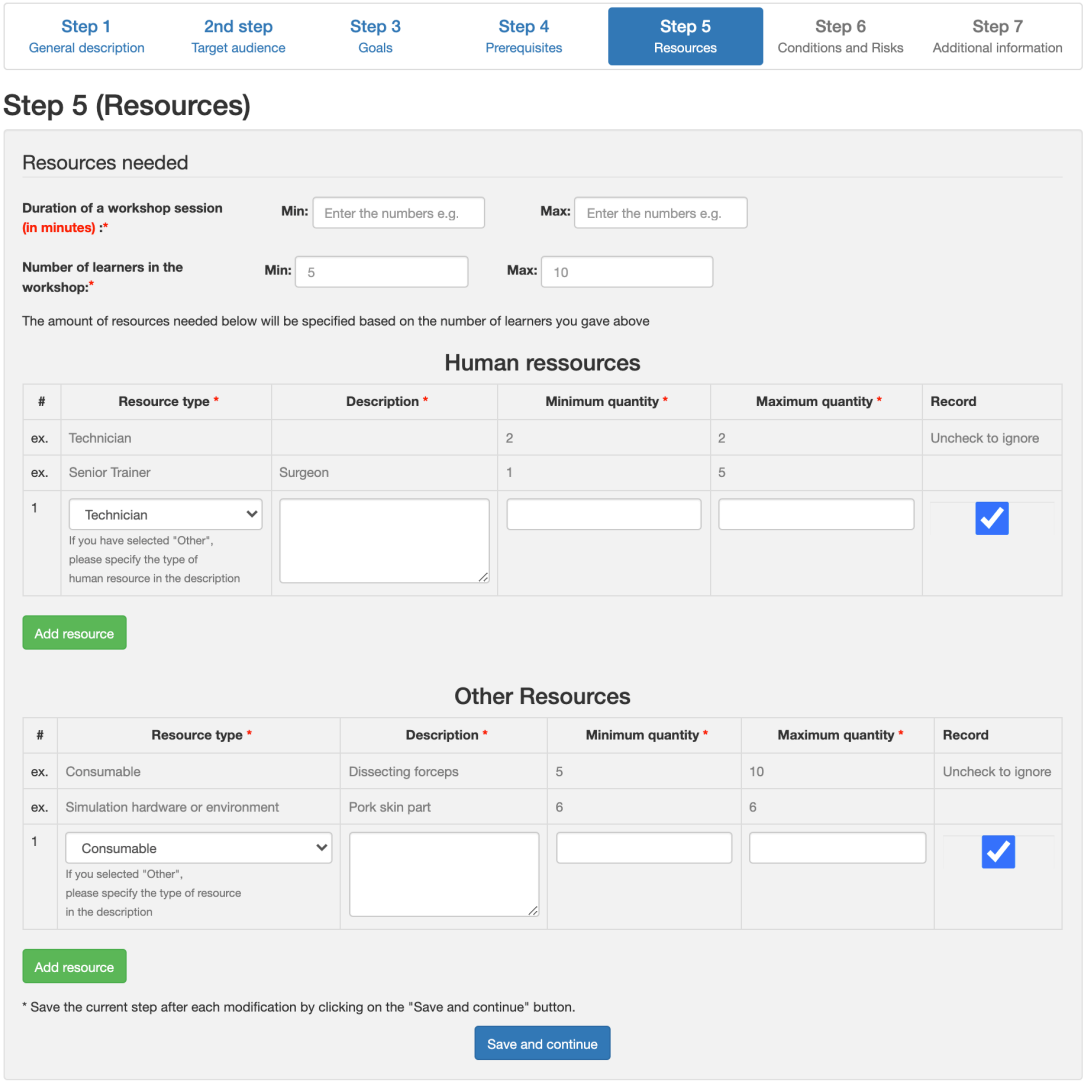

FIGURE 3.3: A part of disseminated questionnaire to educe the required resources.

representing a 25% return rate nationwide. Table 3.1 depicts the distribution of answers in three categories of complete, incomplete, and invalid answers. Experts who answered more than 40% of the questionnaire count as "complete", otherwise "incomplete". Also, the experts with irrelevant answers count as "invalid". Moreover, 60.25% of the experts declared that they want to continue cooperating with the project.

TABLE 3.1: Distribution of domain experts as a function of questionnaire responses.

	Agreed for future collaboration	**Not agreed for future collaboration**
Nb of complete answers	55.00 %	6.00%
Nb of incomplete answers	5.25%	22.25%
Invalid answers	11.50%	

To provide ample expressivity for the experts, we let most answers in the subjective parts of the questionnaire to be of type free text. The advantage of this design choice is that the experts do not go through the burden of selecting options in large lists, and

hence they are motivated to enter their knowledge in whatever form they want. The inconvenience, however, is that the answers are not unified, as the experts may use different terminology or phrasing structure to describe their knowledge. To alleviate this irregularity, we employ two simple yet effective text mining approaches, namely "noun phrase extraction" and "part-of-speech (POS) tagging" [Lor18], to find commonalities between the answers. First, by employing POS tags, we pruned unnecessary and inefficacious parts of the responses, e.g., parts tagged as coordinating conjunctions ("and", "or", "for", etc.) and modals ("could", "will", etc.). Then, the retrieval of noun phrases allowed us to focus on the most important aspects of the answers.

By employing the aforementioned text mining approaches over the results of the online questionnaire, we grouped the answers per each simulation unit. As a result, we identified 83 different simulation learning units, within which 30 were described frequently by the different professionals, and the rest were either entered only once or described partially.

3.3.3 Enhancement

We improved and refined the core OntoSAMSEI by modeling commonalities between the simulation units described by different professionals in different parts of the questionnaire. 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, sub-properties, or instances. We added these simulation sessions to the hierarchy of simulation learning unit as a descendant of the principled class. The enhanced structure of OntoSAMSEI using the results of the online questionnaire reached 470 classes, 49 properties, 550 instances, and 700 constraints. Below, we show several extractions of different parts of the resulting ontology. The full declaration of OntoSAMSEI ontology is accessible via the following link: *https://data.world/baghersh/ontosamsei*.

OntoSAMSEI is modeled using an ontology editor and visual modeling environment called TopBraid Composer [PCHK20]. It is a tool for creating and managing domain models and ontologies in the Semantic Web standards, such as RDF, RDFS and OWL. Each class, property, and instance is identified with a URI (Uniform Resource Identifier) and has a label in both French and English languages. To increase readability, we employ the namespace prefix `samsei` instead of the whole namespace URIs.

Figure 3.4 illustrates a part of the specialization hierarchies of properties and classes resulting from RDFS ontological constraints declared in OntoSAMSEI. We highlight additional classes and properties compared to the hierarchy of classes and properties in the first step (Figure 3.2). We observe there has been ample enhancement to the following

classes: `samsei:Material`, `samsei:Simulator`, and `samsei:Training`. Also there exist 26 added classes to the simulation learning units, such as `samsei:Echocardiography`, `samsei:Intubation`, and `samsei:RoboticSurgery`. Moreover, the enhancement process has resulted in the creation of new classes, i.e., `samsei:Knowledge` (the knowledge bearing the specialities of the learning units) and `samsei:Content` (docimological and pedagogical contents, e.g., courses, medias, MOOCs[3], and serious games).

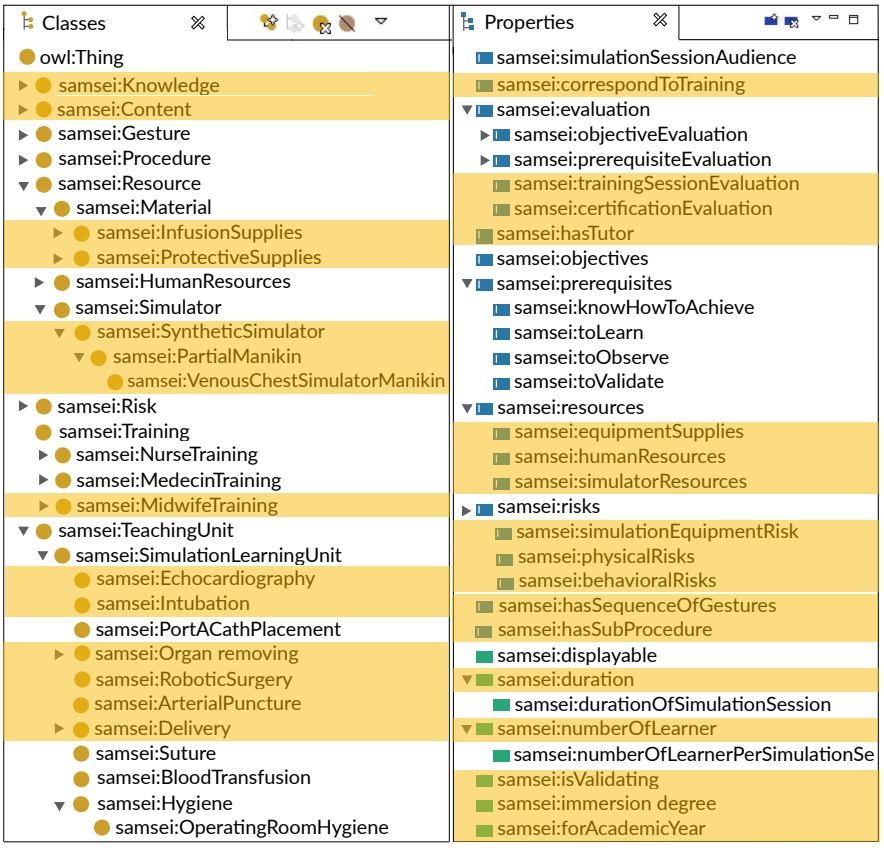

FIGURE 3.4: A part of the hierarchy of properties and classes in the ONTOSAMSEI ontology, visualized in TOPBRAID COMPOSER.

We also note that there exist properties which specialized several other properties, such as the property `samsei:resources`, specialized in `samsei:equipmentSupplies`, `samsei:simulatorResources`, and `samsei:humanResources`. As an example, we focus on a `samsei:SimulationLearningUnit` sub-class called `samsei:Suture` to describe the added properties. The properties `samsei:simulationEquipmentRisk`, `samsei:physicalRisks`, and `samsei:behavioralRisks` are added as new sub-properties of the `samsei:risks`. For instance for `samsei:Suture`, a `samsei:simulationEquipmentRisk` could be "injury caused by needle". The property `samsei:trainingSessionEvaluation` is added as a new sub-property of the `samsei:evaluation` property to define the evaluation of the participants for the learning unit.

[3] *Massive open online course*

The property `samsei:certificationEvaluation` is another new sub-property of the `samsei:evaluation` property which defines the type of the final assessment for a learning unit. The properties `samsei:numberOfLearner` and `samsei:duration` are also new properties to define the number of learners in the learning unit, and the duration of the unit, respectively.

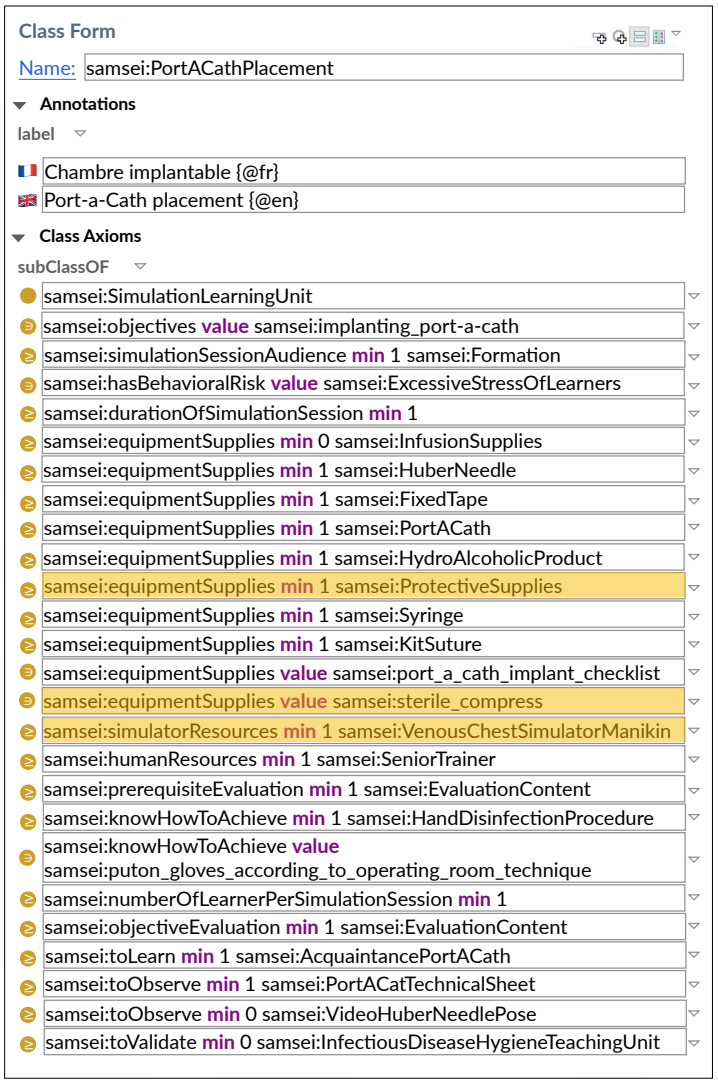

FIGURE 3.5: A part of the hierarchy of OWL constraints for the class `samsei:PortACathPlacement` in ONTOSAMSEI, visualized in TOPBRAID COMPOSER.

Figure 3.5 demonstrates a part of the OWL ontological constraints declared in ONTOSAMSEI for the class `samsei:PortACathPlacement` which is a particular type of simulation learning unit which educates students for placing a port or a catheter. The demonstration is in the syntax proposed by TOPBRAID editor. The OWL ontological constraints declared for this class are `OWL:hasValue` and `OWL:minCardinality`. Note that among the `OWL:minCardinality` constraints, the "minCardinality 0" constraint is superfluous in the OWL semantics. However, it is beneficial to enforce the "optional semantics" and enhance the model readability.

Figures 3.6 and 3.7 provide the RDF graphs corresponding to the following three ontological constraints (highlighted in Figure 3.5):

- `samsei:equipmentSupplies value samsei:sterlie_compress`;
- `samsei:equipmentSupplies min 1 samsei:ProtectiveSupplies`;
- and `samsei:simulatorResources min 1 samsei:VenousChestSimulator-Manikin`.

These constraints are declared for two properties describing the required resources for conducting this type of simulation-based training session, namely `samsei:equipment-Supplies` and `samsei:simulatorResources`, which are specializations of the property `samsei:resources`.

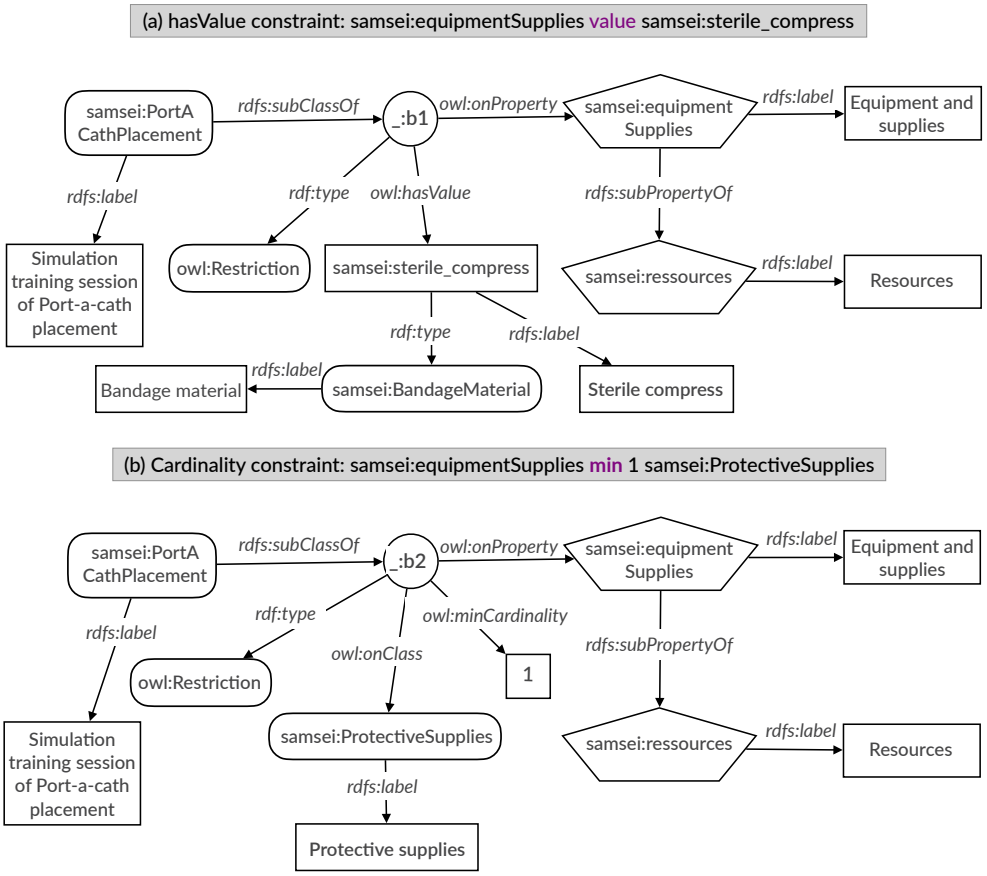

FIGURE 3.6: Two RDF constraint graphs (a) and (b) on the property `samsei:equipment-Supplies` for the class `samsei:PortACathPlacement`.

Figure 3.6 visualize the RDF graphs associated to two first aforementioned constraints on the property `samsei:equipmentSupplies`. The RDF graph in Figure 3.6(a) expresses that `samsei:sterilecompress` (which is an instance of Bandage material) is

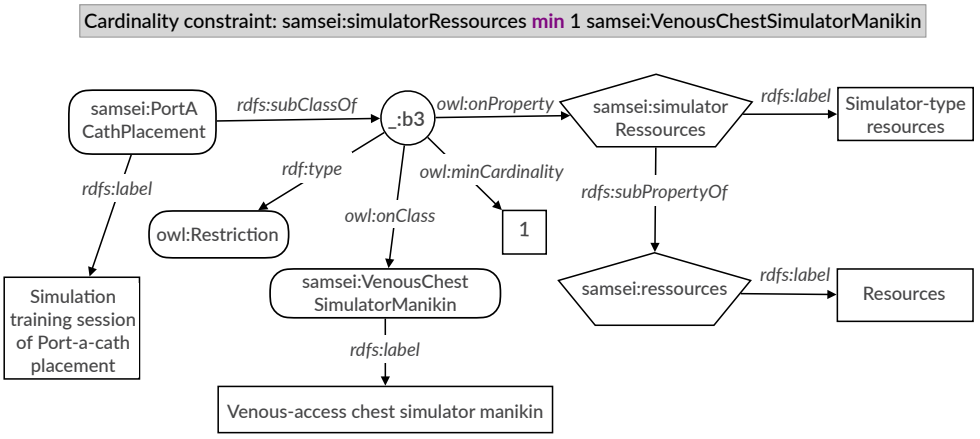

FIGURE 3.7: A RDF constraint graph on the property `samsei:simulator Resources` for the class `samsei:PortACathPlacement`.

declared in the ontology as a mandatory value of the property `samsei:equipmentSupplies`. The RDF graph depicted in Figure 3.6(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.

Figure 3.7 shows the constraint graph associated to the third aforementioned cardinality constraint for the property `samsei:simulatorResources`. The constraint expresses that at least one simulator of type `samsei:VenousChestSimulatorManikin` is mandatory to educate students to place a port or a catheter on the right spot of the patient body.

3.3.4 Ontology Update

Beyond ontology engineering for OntoSAMSEI, our methodology assumes that the ontology is evolving, and not static. Therefore, we design and implement an interactive approach for updating ontologies by involving users in the process. We handle updates automatically through a few interactions with the expert, using a Graphical User Interface, named IOPE. The targeted users are domain experts, who are by default inexperienced in ontology formalization and engineering. In particular, they may not be familiar with the RDF format and the machinery underlying the different components of an ontology. This expert-in-the-loop approach enables the domain experts to enrich the current ontology interactively, by adding new classes and/or constraints, and also guides the less experienced experts in adding new instances to the existing simulation-based training units.

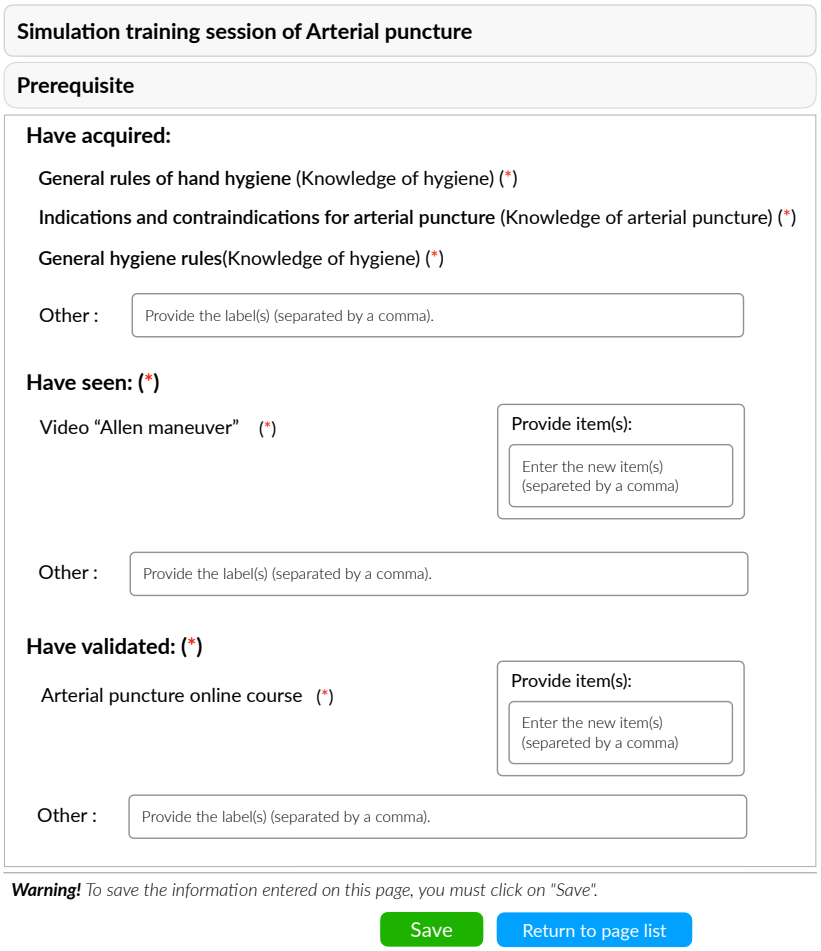

FIGURE 3.8: HTML Web page generated from the ONTOSAMSEI ontology.

As an example of this interactive process, Figure 3.8 illustrates a page which is automatically constructed from ONTOSAMSEI ontology to indicate the prerequisites of the simulation learning sessions for performing arterial puncture[4] on the patient body. These prerequisites consist of acquiring general rules of hygiene and indications and contraindications for the process of arterial puncture. They also indicate the contents that the trainees should observe before attending this training session, i.e., watching the pedagogical video of "Allen maneuver"[5], and validating the arterial puncture online course. Chapter 4 will provide details about the IOPE framework.

[4] *http://www.lumen.luc.edu/lumen/MedEd/medicine/pulmonar/procedur/artstep1.htm*
[5] *https://www.physio-pedia.com/The_Allen_Test_for_Blood_Flow*

3.4 Summary

In this chapter, we presented the first contribution of this thesis, i.e., a methodology for designing and engineering a simulation-based medical training ontology, called OntoSAMSEI ontology. We categorized the related work on ontology engineering methodology (OEM) into three groups of non-collaborative, collaborative, and custom OEMs. We detailed the literature on collaborative OEMs, as they are more adapted to ill-defined domains (domains where formal models are lacking, and documentations are scarce), and proposed our own approach as a mixture of DILIGENT [VPTS05] and UPON-Lite [DNM16], two collaborative OEMs.

Our approach is a 4-step process. First, it begins by *ontology bootstrapping* (i.e., build a small initial ontology with the help of domain experts), followed by *knowledge elicitation* using online questionnaires to fill in the ontology. The third step is *enhancement*, where we improve and refine the core ontology by modeling commonalities between the units described by different professionals in different parts of the questionnaire. The resulting ontology is a hierarchy of classes and of properties, enriched by *ontological constraints* on the properties and on the classes. In the last step, *ontology update*, our approach goes beyond ontology engineering to account for the evolving nature of the ontology. For this aim, we design an interactive expert-in-the-loop approach for updating ontologies by involving experts in the process. This approach is the subject of our second contribution, detailed in Chapter 4.

Chapter 4

Interactive Ontology Population and Enrichment

4.1 Introduction

"The only constant in life is change"[1], and ontologies are not exceptions. By their very nature, real world ontologies are dynamic artifacts that evolve copiously both in their structure (i.e., data model) and their content (i.e., instances). Keeping them up-to-date is hence 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, using 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. A major source of inefficiency is that "schema modeling" and "data creation" are often inseparable (and indistinguishable by most novice experts) in the process of ontology updates, hence smooth and lightweight updates attenuate.

In this chapter, we advocate for an alternative and more effective data publishing approach, and propose to handle updates automatically through a few interactions with the expert, using a Graphical User Interface (GUI). This GUI operates on top of the enhancement layer discussed in Chapter 3 to complete the loop of ontology engineering and enable fruitful interactions with the user.

[1] *Famous quote by Heraclitus, ancient Greek philosopher*

The challenges associated to interaction-based automatic updates are two-fold:

- 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 (which presumably consist of thousands of entities and millions of facts) for the consumption of experts. While there exist several methods to visualize a graph structure [BBDW17, HFM07, FCL+17], the outcome is often hard to digest by domain experts.
- 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.

We have designed IOPE (Interactive Ontology Population and Enrichment), a framework for the automatic construction of a Graphical User Interface (GUI) consisting of a set of *Web pages* that are *pre-filled* based on the ontological constraints present in an input ontology. We leverage Web pages as a natural interaction means to tackle the challenge of counter-intuitive ontology representations. IOPE generates and pre-fills the Web pages from ontological constraints, which supports the controlled update process of a given ontology. We illustrate the generation process of IOPE using some examples on the OntoSAMSEI ontology. However, as IOPE is generic and can be applied to ontologies from a variety of domains, Chapter 5 contains evaluations of the effectiveness of our approach on other specialized ontologies as well.

This chapter is organized as follows: Section 4.2 presents the literature on ontology update and automatic construction of forms. In Section 4.3, we present our methodology for the automatic construction of a GUI from an input ontology, and its application for guiding the ontology updates (population and enrichment). Last, we summarize this chapter in Section 4.4.

4.2 Related Work

To the best of our knowledge, no approach in the literature has proposed and/or formalized an interactive system for enrichment and population of specialized ontologies using ontological constraints. However, our work does relate to a number of others in functionality and applicability. We discuss the related work about ontology editing tools (Section 4.2.1) and graph-based and form-based ontology update (Sections 4.2.2 and 4.2.3, respectively).

4.2.1 Ontology Editing Tools

Various methods are proposed in the literature to update the structure (i.e., enrichment) and the content (i.e., population), via interactions with domain experts [NSD+01, PCHK20, MPE+15]. A vast majority of these methods offer solely one of the enrichment or population updates.

In the literature, ontological updates are often performed using ontology editing tools, such as Stanford's Protégé [NSD+01], TopQuadrant's TopBraid [PCHK20], and Metaphacts' Ontodia [MPE+15]. However, these systems require a basic understanding of the RDF notation and of the OWL semantics to edit the ontology consistently. Moreover, users must be able to recognize and correctly encode the expected property relationships for each new class instance according to RDF(S)/OWL semantics. Second, the list-based organization of the ontology (e.g., see Figure 3.5 for TopBraid) does not represent the relations in data intuitively. Last, as the schema and the data are presented in the same place, the experts have often difficulties distinguishing between the two, and are consequently prone to make unintentional erroneous edits.

4.2.2 Graph-based Ontology Update

Graph-based editing approaches alleviate some of the limitations in the status quo of ontology editing tools, by leveraging shapes graphs in the form of SHACL standard[2] [WMH+20, VPCAR20]. While shapes graphs are well adapted for editing complex data, they require the definition of such graphs for each ontology, independently. 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 [WLA18] is a web application for the interactive graph-based visualization of ontologies which employs the Visual Notation for OWL Ontologies (abbr., VOWL) [LNHE16]. While VOWL contributes to standardizing and generalizing OWL visualization, 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 pages as a more widespread medium for visualizing information. IOPE has also an extended support for the update of instances and of ontological constraints.

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

4.2.3 Form-based Ontology Update

Forms are used in [MFC+17] in a nested structure to capture relational aspects of knowledge graphs and update RDF data. The experts are also guided with dynamic suggestions based on existing data. However, the nested structure introduces increasing complexity and hence lacks intuitiveness. Moreover, the focus in [MFC+17] is solely on the population part and the approach does not extend to OWL constraints.

In [BHLX13], 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.

4.3 IOPE Approach

Our approach consists of 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 then transformed into RDF triples that must be verified by an ontology engineer before being permanently added in the domain ontology. Figure 4.1 provides an overview of IOPE's workflow.

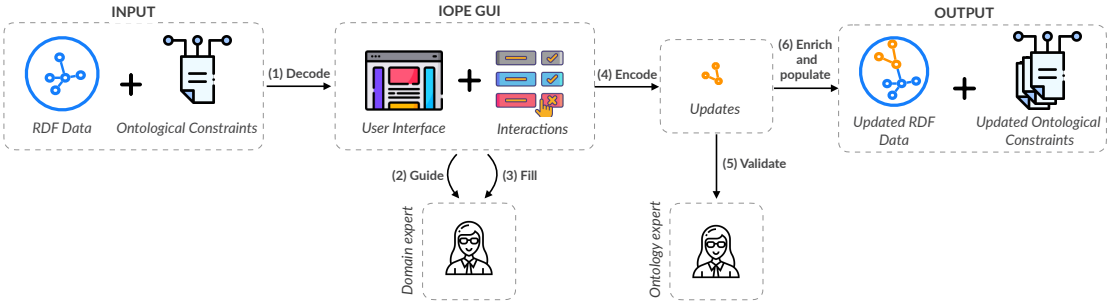

FIGURE 4.1: The overview of IOPE's workflow.

The IOPE GUI is made of Web pages that are automatically generated and pre-filled to reflect the domain ontological constraints. For the generation of the pre-filled Web pages, we follow a declarative approach based on a set of *mapping rules* from *RDF constraint graphs* to *Web form templates*. The Web form templates are described using a Web form

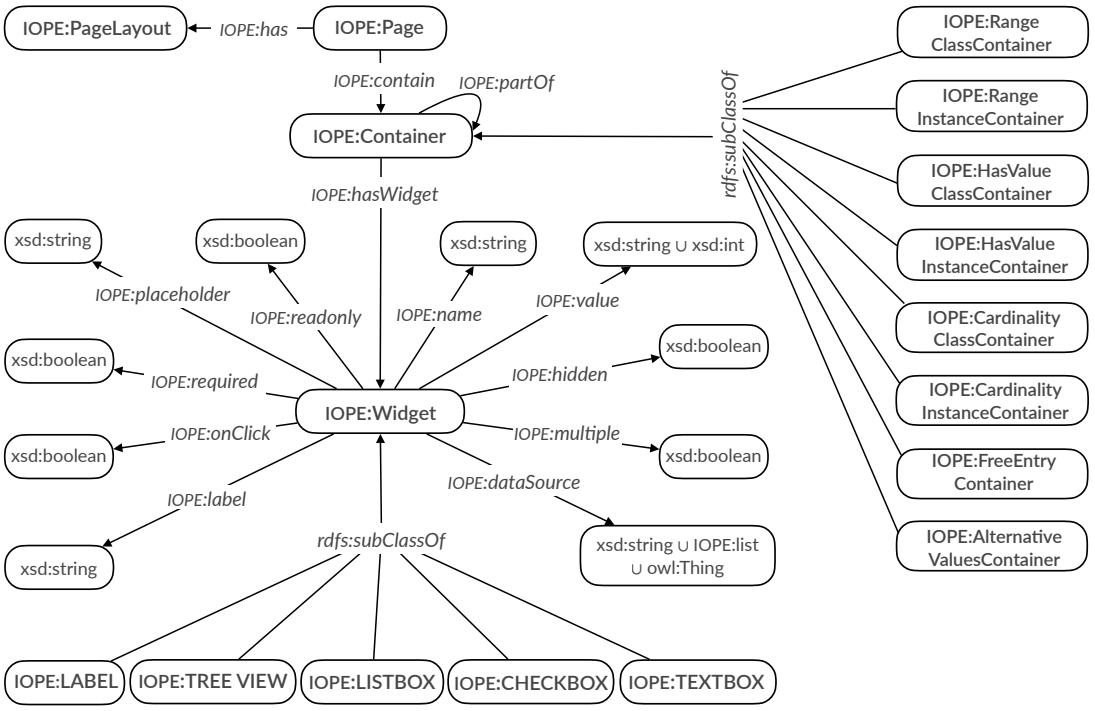

FIGURE 4.2: The IOPEWEB form ontology

ontology called IOPEWEB, which we developed by adapting RaUL ontology [HUH10] to our context. We explain the IOPEWEB ontology in Section 4.3.1, followed by a discussion over the mapping rules in Section 4.3.2. The input of the expert on the pre-filled Web pages needs to be bound to RDF data using a set of *binding rules*. We also explain the data binding mechanism in Section 4.3.3.

4.3.1 The IOPEWeb Ontology

The IOPEWEB ontology is shown in Figure 4.2. The ontology is organized around four 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.

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.

Moreover, IOPEWeb considers some additional properties to describe the functionality of widgets, as mentioned below.

- Data type properties: `IOPE:label` (a description attached to the widget), `IOPE:name` (identifier of the widget), `IOPE:placeholder` (a by-default string value which provides a hint for the value to fill the widget with);
- Boolean properties: `IOPE:hidden` (whether the widget is invisible to the user or not), `IOPE:multiple` (whether the widget can accept multiple values), `IOPE:readonly` (whether the widget is modifiable) and `IOPE:onclick` (whether the widget is clickable), `IOPE:required` (if set to True, the widget will be rendered by a red asterisk, to specify that it must be filled in by the user.)

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. Examples are `IOPE:RangeClassContainer`, `IOPE:RangeInstanceContainer`, `IOPE:HasValueClassContainer`, and others expressed on the right side of Figure 4.2. In Section 4.3.2, we explain the relation between the container sub-classes and the ontological constraints.

The ordering of the widget elements in the containers and in the Web pages are defined within a page layout. Figure 4.3 shows the empty page layout that is generated at the initialization process of building the IOPE GUI for a given focus class F, i.e., the class chosen by the user as her class of interest.

4.3.2 Ontology-based GUI Construction

In a declarative approach, we employ a set of *mapping rules* to generate pre-filled Web pages, in an automated fashion. The input required for GUI construction is a domain ontology in which the ontological constraints are automatically saturated by a reasoning algorithm. In this section, first we discuss the initialization and saturation process (Section 4.3.2.1), and then present the mapping rules to construct the ontology-based GUI (Section 4.3.2.2).

4.3.2.1 Initialization

Given the ontological constraints that we covered in Chapter 2, the saturation of the constraints can be done iteratively through a breadth-first traversal of the class hierarchy, as explained below:

- For each class D, compute the set *Constraints*(D) of all constraints holding for D, by adding the ontological constraints declared for D's super-classes to the set of ontological constraints declared for the class.
- Simplify the resulting set of ontological constraints by removing the redundant ones. A constraint `(p min k c)` in *Constraints*(D) is considered redundant if there exists a constraint `(sp min k' sc)` in *Constraints*(D) 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 vis-à-vis the (declared) constraint `(samsei:equipmentSupplies min 2 samsei:Syringe)` within a given set of constraints.

Once the redundant-free set *Constraints*(D) is computed, 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), where P is a property involved in at least 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).
- Otherwise, *Group*(P, F) is the subset of all the constraints involving the sub-properties of P.

In the following, we focus on the more general case, i.e., the "otherwise" condition. For each group of properties *Group*(P, F), an instance of a Web page is created with the page layout depicted in Figure 4.3. The page layout defines the organization of the Web page with a set of specific containers dedicated to different ontological constraints. Given *Group*(P, F), the constraints on sub-properties p of P will be positioned in their dedicated containers. We will now explain the connection between the containers and the ontological constraints.

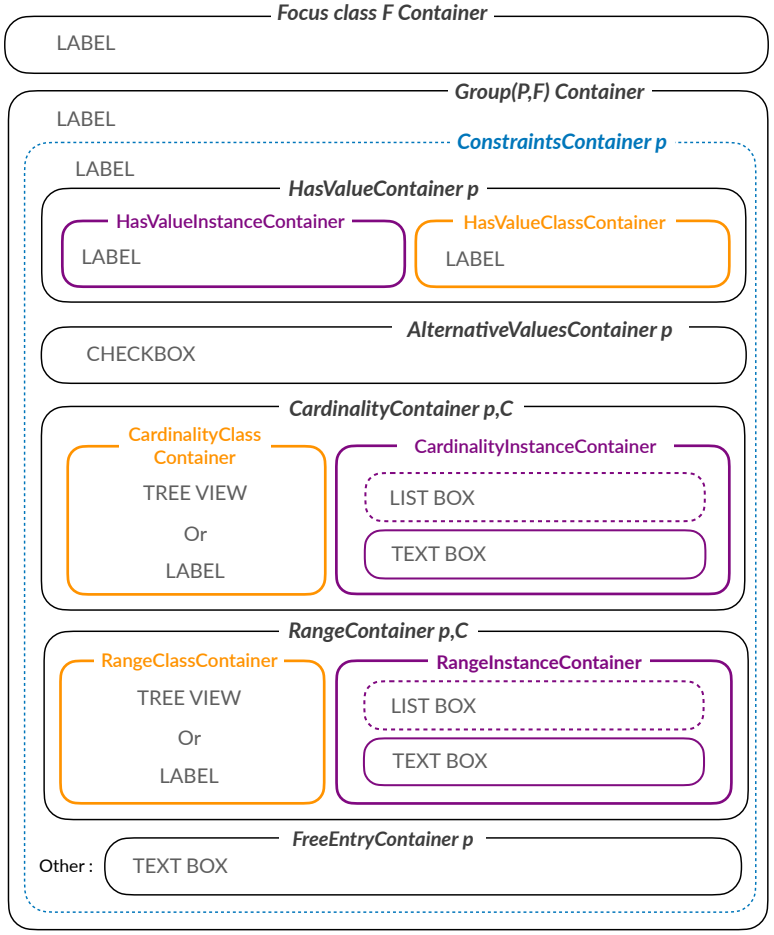

FIGURE 4.3: Web page template prepared for the rendering of constraints of the focus class F for each property p which is a specialization of a same property P.

The following instances of the `IOPE:Container` class are created, with their pre-allocated positions in the Web page template shown in Figure 4.3, which is initialized as empty.

- "`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.

Next, we explain how 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.

4.3.2.2 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 in the left-hand side expresses a particular ontological constraint on a property and a (focus) class, and the IOPEWeb graph pattern in the right-hand side 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. The implementation of the mapping rules is implemented using RDFLib[3] and JSON[4] libraries in Python 2.7.16. Our implementation is publicly available in [BT21].

For clarity purposes, we describe the mapping rules in their instantiated form. There are 16 mapping rules. To avoid redundancy, we describe 9 mapping rules in this section, and present the rest in Appendix B. To distinguish between individual and grouped mapping rules, we mark the former with a "■" symbol, and the latter with a "▶".

■ **Mapping rule #1 for a focus class F on the sub-properties p of a property P.** This mapping rule is presented in Figure 4.4. The rule applies for each group $Group(P, F)$ of properties, and each property p in $Group(P, F)$ as follows:

[3] *https://rdflib.readthedocs.io/en/stable/*
[4] *https://docs.python.org/3/library/json.html*

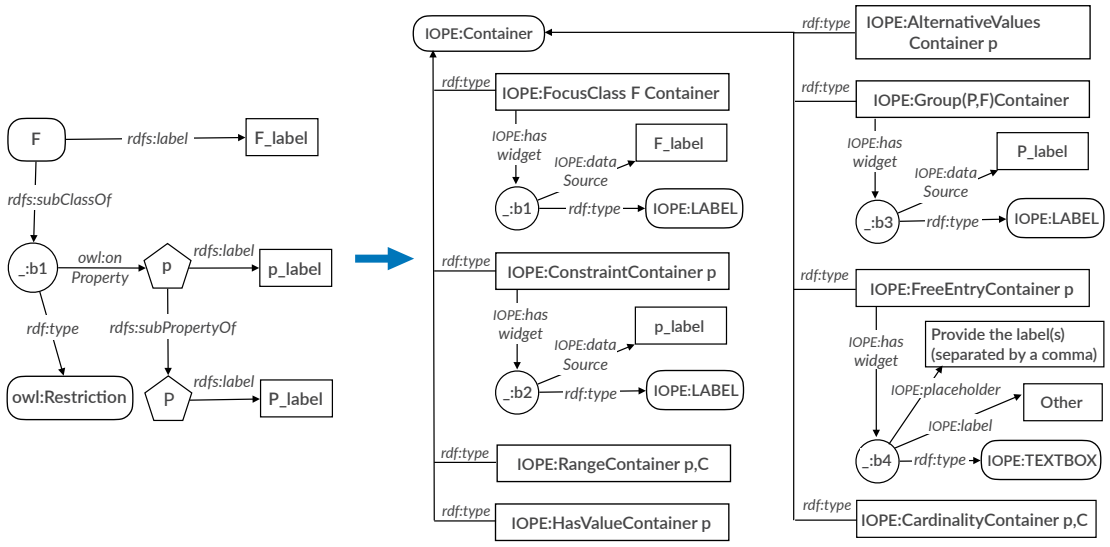

FIGURE 4.4: Mapping rule #1, employed for a focus class F on the sub-properties p of a property P.

- The specific containers "`IOPE:FocusClass` F `Container`", "`IOPE:` $Group(P, F)$ `Container`", and "`IOPE:ConstraintContainer` p" are declared as instances of the `IOPE:Container` class, and widgets 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 "`IOPE:FreeEntryContainer` p" is declared as an instance of the `IOPE:Container` class, where p is an object property. The associated widget of type `IOPE:TEXTBOX` is created as a blank node, to collect user future inputs over the property p (in the form of user interactions on p), i.e., an ontology enrichment task.

- The four specific containers "`IOPE:HasValueContainer` p", "`IOPE:AlternativeValuesContainer` p", "`IOPE:RangeContainer` p, C", and "`IOPE:CardinalityContainter` p, C" are declared as instances of the `IOPE:Container` class. The widgets associated to them are created by other mapping rules, which we describe next.

■ **Mapping rule #2 for a value restriction** (`p value v`) **for** F **such that** (`v rdf:type C`)**.** This mapping rule is presented in Figure 4.5. The specific container "`IOPE:HasValueContainer` p" is decomposed into two sub-containers defined as blank nodes, whose types are `IOPE:HasValueInstanceContainer` and `IOPE:Has-ValueClassContainer`. For the two sub-containers, widgets of type `IOPE:LABEL` are created as blank nodes with the property `IOPE:dataSource` filled

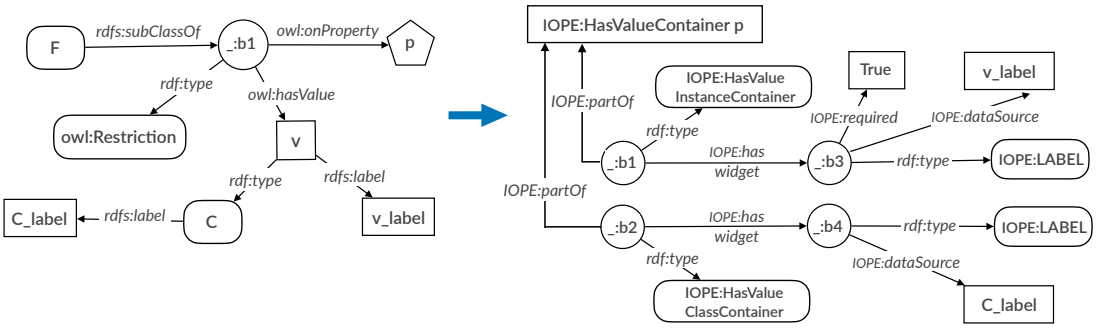

FIGURE 4.5: Mapping rule #2, employed for a value restriction (*p* `value` *v*) for *F* such that (*v* `rdf:type` *C*).

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.

▶ **Mapping rules #3 to #7 for a cardinality restriction** (*p* `min` *n C*) **for** *F* **such that** $n > 0$. This group consists of five rules where each counts as an independent mapping rule.

■ **Mapping rule #3** corresponds to the case where C has a hierarchy of sub-classes and a list of instances in the domain ontology. This mapping rule is presented in Figure 4.6. The specific container "`IOPE:CardinalyContainer` p, C" is decomposed into two sub-containers defined as blank nodes, with types "`IOPE:CardinalityClassContainer`" and "`IOPE:CardinalityInstanceContainer`".

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 (if any).

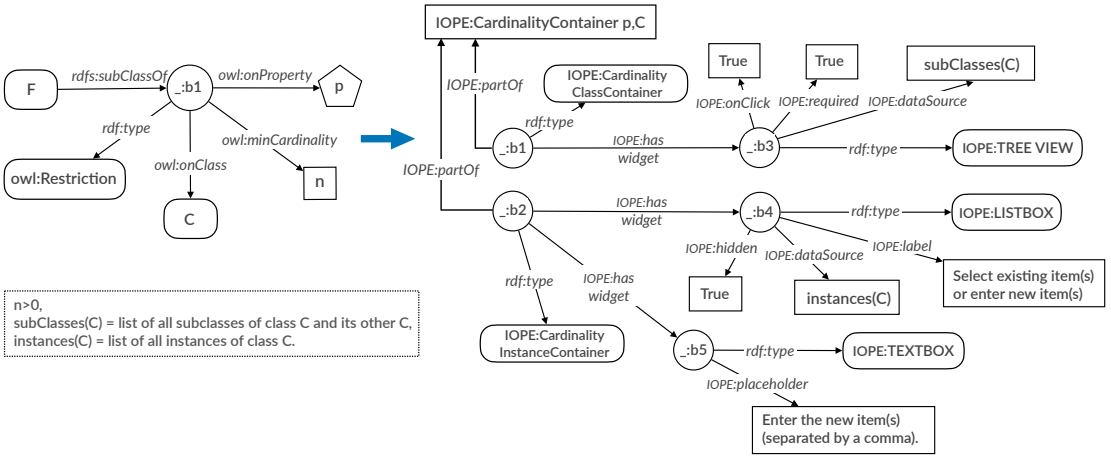

FIGURE 4.6: Mapping rule #3, employed for a `cardinality` constraint where $subClasses(C)$ and $instances(C)$ are not empty, and $n > 0$.

■ **Mapping rule #4** corresponds to the case where C does not have either a hierarchy of sub-classes or a list of instances in the domain ontology. This mapping rule is presented in Figure 4.7. For `IOPE:CardinalityClassContainer`, a widget of type `IOPE:LABEL` is created as a blank node with the property `IOPE:dataSource` filled by the label of class C, i.e., "*C_label*". The property `IOPE:required` is set to True for this widget to indicate that this value is mandatory for the property p. As the class C does not have any instances for `IOPE:CardinalityInstanceContainer`, a widget of the type `IOPE:TEXTBOX` is created with the `IOPE:placeholder` property set to the value "enter the new item(s) (separated by a comma) or provide a minimal number of items" in order to enable users to enter new instances, or provide a minimum number of requirements.

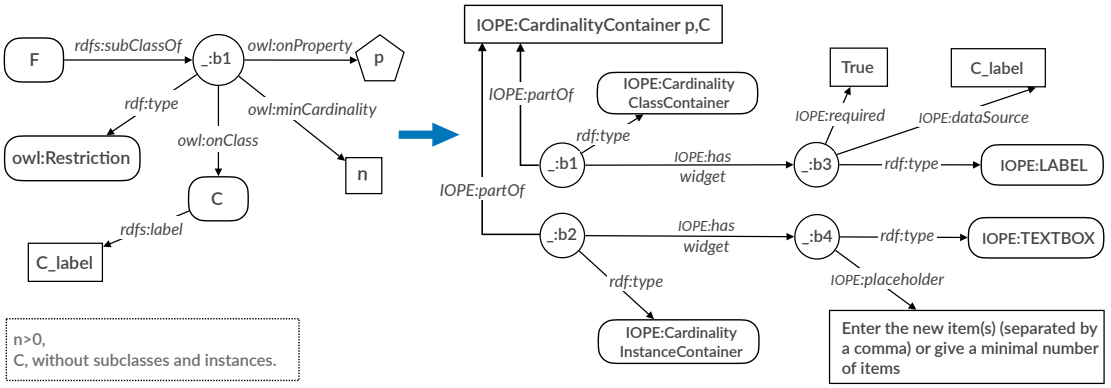

FIGURE 4.7: Mapping rule #4, employed for a `cardinality` constraint where both $subClasses(C)$ and $instances(C)$ are empty, and $n > 0$.

■ **Mapping rule #5** corresponds to the case where C does not have a hierarchy of sub-classes, but has a list of instances in the domain ontology, and **Mapping rule #6** corresponds to the case where C has a hierarchy of sub-classes but does not have a list

of instances in the domain ontology. These two mapping rules are variants of the two previous rules, and are described in Appendix B.

■ **Mapping rule #7** corresponds to the case where p is a datatype property. This mapping rule is described in Figure 4.8. For `IOPE:CardinalityInstanceContainer`, a widget of type `IOPE:TEXTBOX` is created with the `IOPE:placeholder` property set to the value "enter a value" in order to enable users to enter the numerical value they need for the property p.

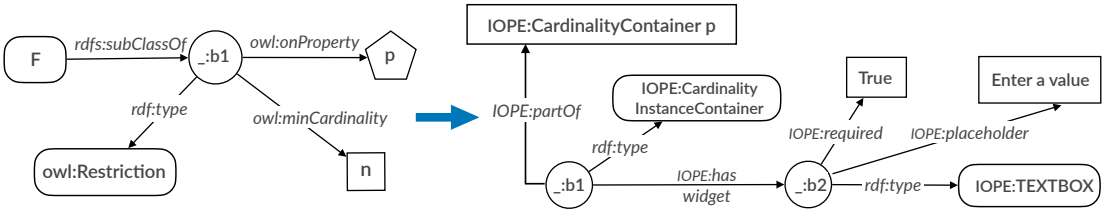

FIGURE 4.8: Mapping rule #7, employed for a `cardinality` constraint where p is a datatype property, and $n > 0$.

► **Mapping rules #8 to #11 for a cardinality restriction** (p `min` n C) **for** F **such that** $n = 0$. This group consists of four rules, where each counts as an independent mapping rule.

■ **Mapping rule #8** corresponds to the case where C has a hierarchy of sub-classes and a list of instances in the domain ontology. This mapping rule is presented in Figure 4.6. The specific container "`IOPE:CardinalyContainer` p, C" is decomposed into two sub-containers defined as blank nodes, with types "`IOPE:CardinalityClassContainer`" and "`IOPE:CardinalityInstanceContainer`".

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`. Given $n = 0$, the property `IOPE:required` is set to False for this widget, to indicate that it is up to the user to select a choice from this widget. The property `IOPE:onClick` is set to True for this widget to indicate that, 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,

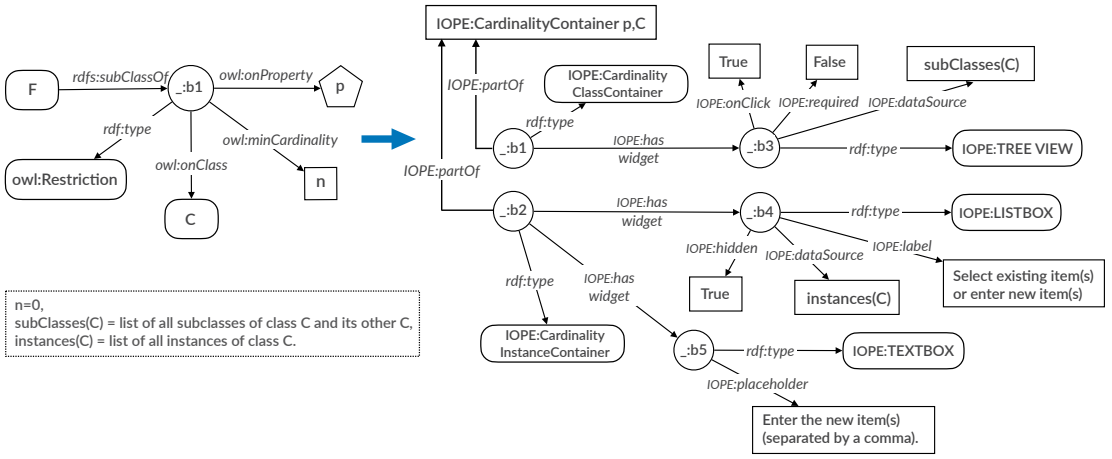

FIGURE 4.9: Mapping rule #8, employed for a `cardinality` constraint where $subClasses(C)$ and $instances(C)$ are not empty, and $n = 0$.

whose value is set to "enter the new item(s) (separated by a comma)", in order to enable the user to enter new instances (if any).

■ **Mapping rule #9** corresponds to the case where C does not have either a hierarchy of sub-classes or a list of instances in the domain ontology, **Mapping rule #10** corresponds to the case where C does not have a hierarchy of sub-classes, but has a list of instances in the domain ontology, and **Mapping rule #11** corresponds to the case where C has a hierarchy of sub-classes but does not have a list of instances in the domain ontology. These three mapping rules are variants of the previous rules, and are described in Appendix B.

► **Mapping rules #12 to #15 for "domain and range" constraints for F on property p such that** (p `rdfs:domain` F) **and** (p `rdfs:range` C). This group consists of four rules where each counts as an independent mapping rule.

■ **Mapping rule #12** corresponds to the case where the class range of the property p has sub-classes and instances. Figure 4.10 presents this mapping rule. In this case, the specific container "`IOPE:RangeContainer` p, C" is decomposed into two sub-containers defined as blank nodes, with types "`IOPE: RangeClassContainer`" and "`IOPE:RangeInstanceContainer`".

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:onClick` are set to True for this widget to indicate that 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 (if any).

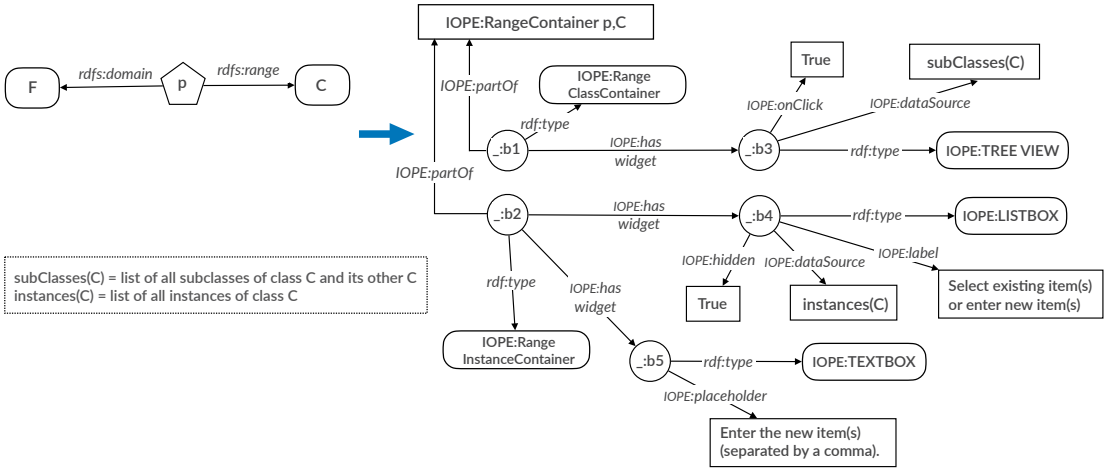

FIGURE 4.10: Mapping rule #12, employed for "domain and range" constraints, where $subClasses(C)$ and $instances(C)$ are not empty.

■ **Mapping rule #13** corresponds to the case where the class range of the property p is without sub-classes and instances. Figure 4.11 presents this mapping rule. In this case, a widget of type `IOPE:LABEL` is created for `IOPE:RangeClassContainer`. The widget is a blank node with the property `IOPE:dataSource` filled by the label of class C, i.e., "*C_label*". As the class C does not have any instances for `IOPE:RangeInstanceContainer`, a widget of the type `IOPE:TEXTBOX` is created with the `IOPE:placeholder` property set to the value "enter the new item(s) (separated by a comma) or provide a minimal number of items" in order to enable users to enter new instances, or provide a minimum number of requirements.

■ **Mapping rule #14** corresponds to the case where the class range of the property p does not have a hierarchy of sub-classes, but has a list of instances in the domain ontology, and **Mapping rule #15** corresponds to the case where the class range of the property p has a hierarchy of sub-classes but does not have a list of instances in the domain ontology. These two mapping rules are variants of the two previous rules, and are described in Appendix B.

■ **Mapping rule #16 for an alternative values constraint** (*C* `owl:oneOf` $[v_1 \dots v_n]$)**.** This mapping rule is presented in Figure 4.12. For the specific container

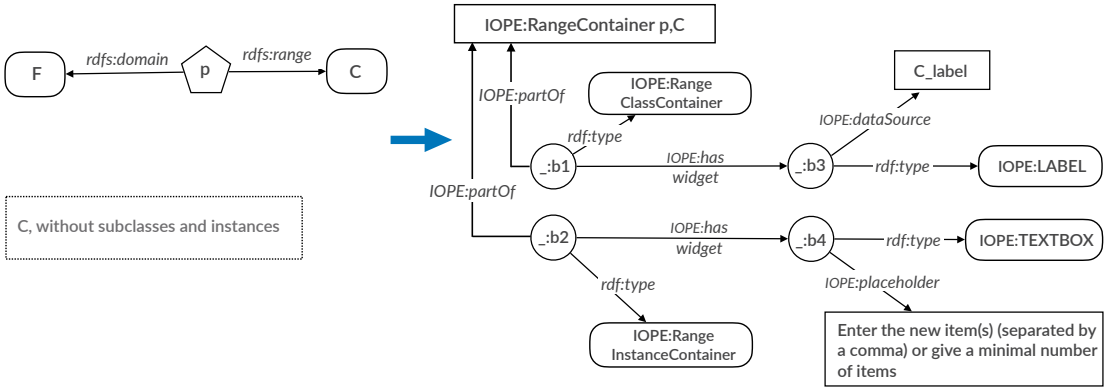

FIGURE 4.11: Mapping rule #13, employed for "domain and range" constraints, where both $subClasses(C)$ and $instances(C)$ are empty.

"`IOPE:Free-EntryContainer` p", n widgets of type `IOPE:CHECKBOX` are created as blank nodes with the property `IOPE:dataSource` filled by the corresponding labels of $v_1 \dots v_n$ from the domain ontology. The property `IOPE:onClick` is set to True for these widgets to indicate that they support the interaction with users.

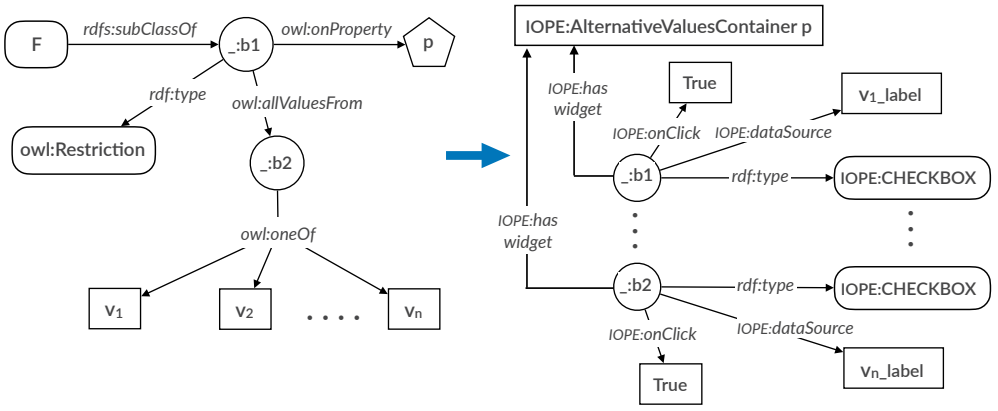

FIGURE 4.12: Mapping rule #16, employed for alternative values constraint.

We proceed with our illustrative example from Section 3.3.3 to elaborate on the aforementioned mapping rules. Figure 4.13 shows the IOPEWEB graph as the outcome of triggering the mapping rules which are applicable to the three constraint graphs displayed in Figure 3.6 (depicting the `HasValue` and `Cardinality` constraint graphs on the property `samsei:equipmentSupplies`) and Figure 3.7 (depicting the `cardinality` constraint graph on the property `samsei:simulatorResources`), both for the focus class `PortACathPlacement`.

In Figure 4.13, the blue color represents the part of IOPEWEB graph which is generated by triggering the mapping rule #1 (Figure 4.4) instantiated appropriately. The violet section in the figure represents the result of triggering the mapping rule #2 (Figure 4.5) for the value restriction (`samsei:equipmentSupplies value samsei:sterile_compress`). The red color represents the results of applying the mapping rule #3

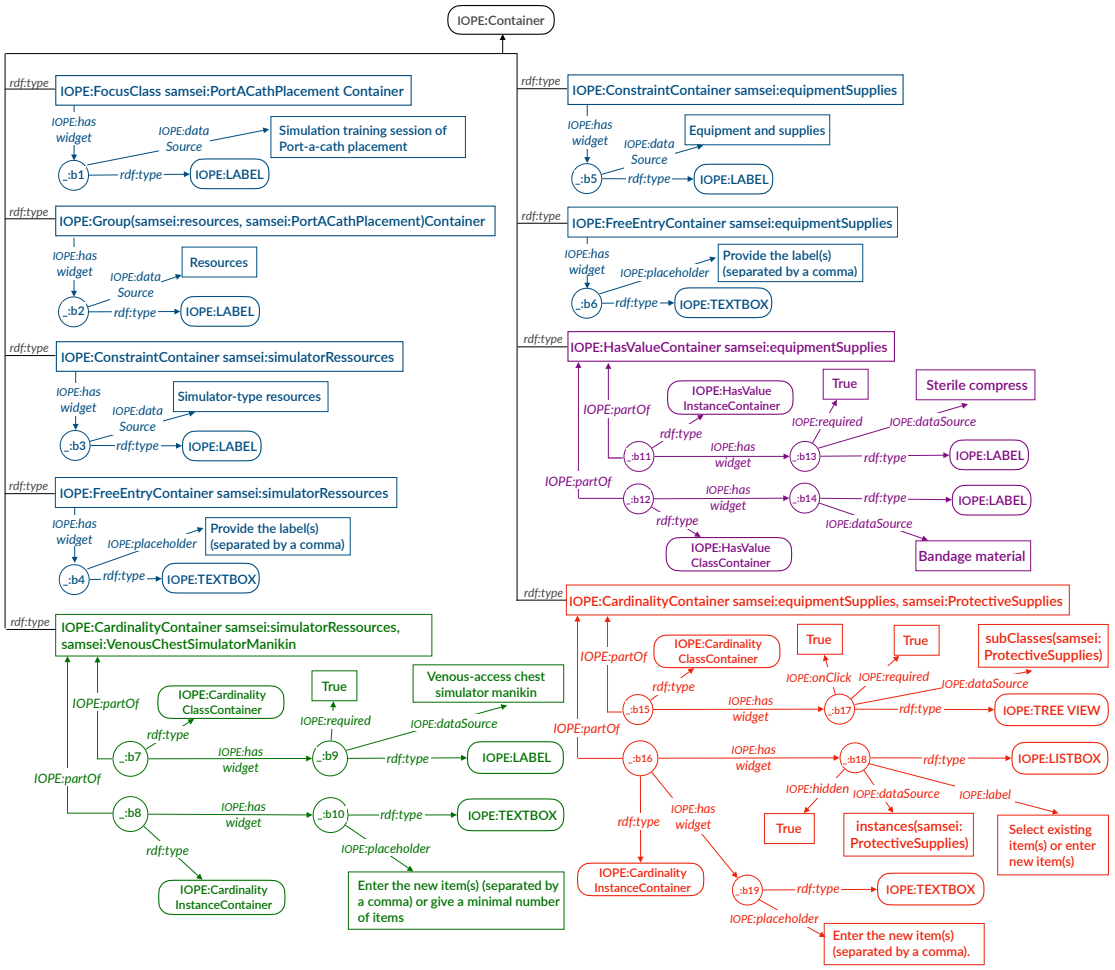

FIGURE 4.13: Illustration of IOPEWEB graph as the outcome of applying the mapping rules on the following example constraints: `(samsei:simulatorResources min 1 samsei:VenousChestSimulatorManikin)`, `(samsei:equipmentSupplies min 1 samsei:ProtectiveSupplies)`, and `(samsei:equipmentSupplies value samsei:sterile_compress)`.

(Figure 4.6) for the cardinality restriction `(min 1 samsei:equipment-Supplies samsei:ProtectiveSupplies)`. The green section corresponds to the application of the mapping rule #4 (Figure 4.7) on the class `samsei:VenousChestSimulator-Manikin` for the following cardinality restriction: `(min 1 samsei:simulatorRes-ources samsei:VenousChestSimulatorManikin)`.

Figure 4.14 left shows the resulting pre-filled Web page generated by the HTML implementation of the IOPEWEB specification. Figure 4.14 right shows the effect of a user interaction through the widget of type the `IOPE:TREEVIEW` to select the sub-class `samsei:DisposableDrape` from `samsei:ProtectiveSupplies` sub-class hierarchy. Note that 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.

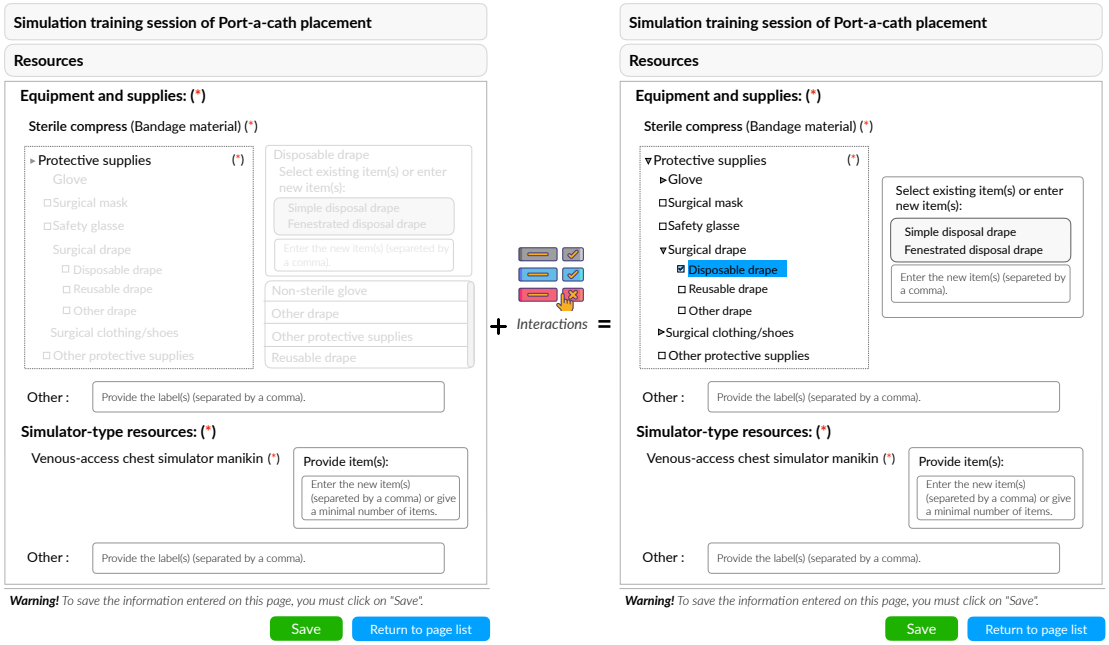

FIGURE 4.14: HTML Web page generated from the application of the mapping rules on the three constraints presented in Figures 3.6 and 3.7 (left), and the evolution of the page after a user interaction through the widget `IOPE:TREEVIEW` (right).

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.

4.3.3 Transforming Interactions to RDF Graphs

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. We define two categories of binding rules:

- **Creation.** The binding rule in this category creates an instance of a focus class in the RDF graph.
- **Value filling.** The second category consists of binding rules which are triggered when the `IOPE:value` property of an input is filled by the user through a widget. Once triggered, the rule enriches the description of the instance already created

using the first-category binding rule. These binding rules can also result in adding new constraints to the focus class.

In the following, we provide the full set of 9 binding rules. The first rule belongs to the first category of binding rules, and the others belong to the second category.

■ **Binding rule #1 (focus class selection).** Figure 4.15 presents this binding rule. The rule 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.

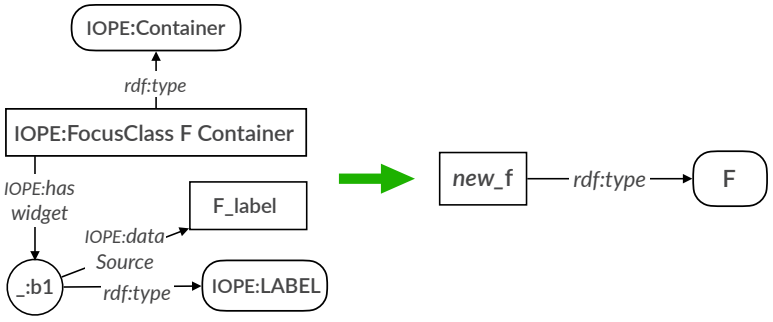

FIGURE 4.15: Binding rule #1, for creating an instance new_f of a focus class F.

■ **Binding rule #2 (textbox filling in free entry container).** Figure 4.16 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 or existing class u and a new `cardinality` constraint for F on the property p. The class u is located as a sub-class of the class D, which is the range of property p. In Figure 4.16 and the rest of the figures for the other binding rules, user interactions are highlighted in red.

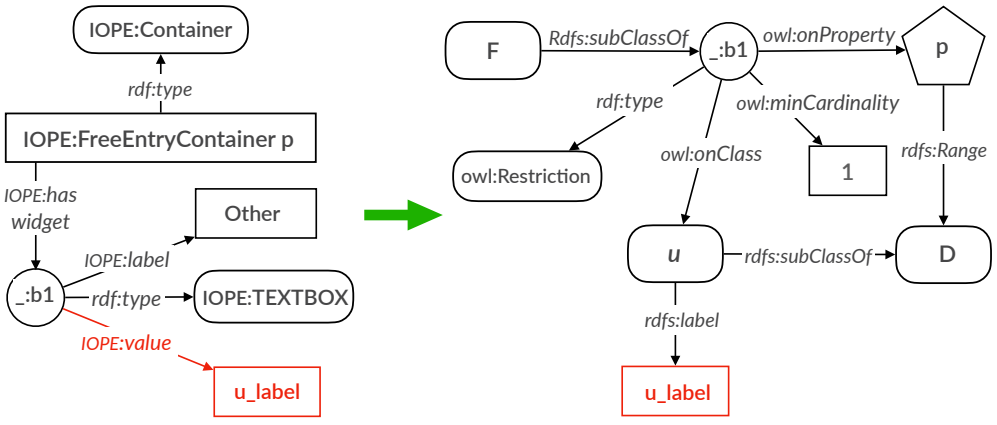

FIGURE 4.16: Binding rule #2, for free entry container on property p and a focus class F.

■ **Binding rule #3 (listbox filling).** Figure 4.17 shows the binding rule for the chosen instance(s) from an `IOPE:LISTBOX` widget related to a selected class in `IOPE:TREEVIEW`

widget. For each class D selected by the user from the IOPE:TREEVIEW widget, the user will also select an instance L from the IOPE:LISTBOX widget. Given D and its instance L, the binding rule creates RDF triples to convey, first, the instance new_f (already created using the binding rule #1) has the instance L via the property p, second, the instance L has the label L_label, and third, L is an instance of D.

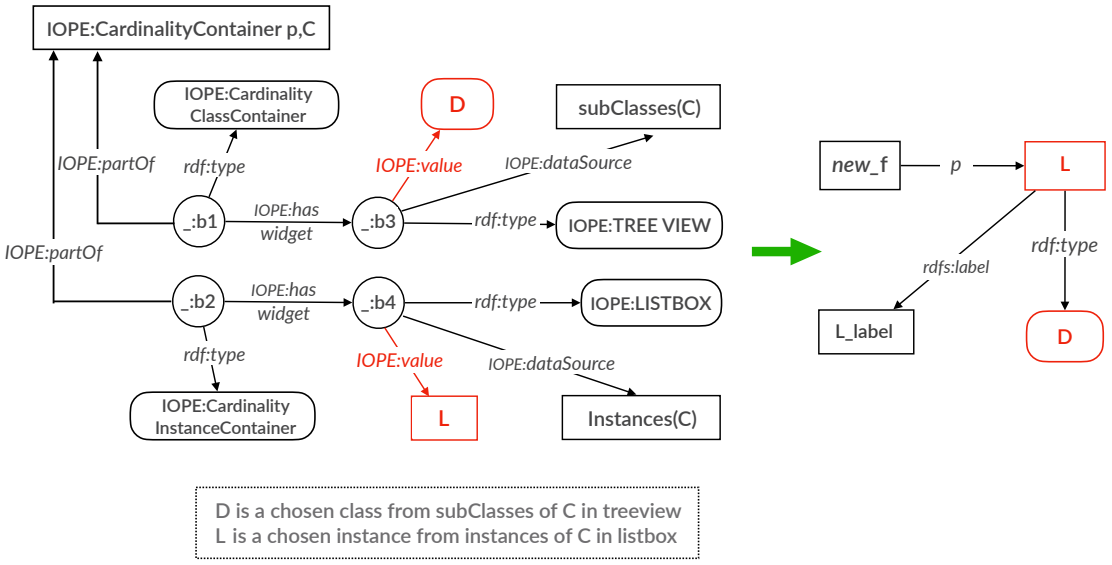

FIGURE 4.17: Binding rule #3, for a selected class D in a IOPE:TREEVIEW and the selected instance(s) L from an IOPE:LISTBOX widget.

■ **Binding rule #4 (textbox filling for a selected class).** Figure 4.18 shows the binding rule for an IOPE:TEXTBOX widget filled by a value of type string related to a selected class in an IOPE:TREEVIEW widget. Given the chosen class D (i.e., a subclass of the class C) in the IOPE:TREEVIEW widget, and the entered item(s) u in the IOPE:TEXTBOX widget, the triggering of this binding rule will generate RDF triples to convey, first, the instance new_f has the value u via the property p, second, the human-readable label of value u is the entered string u_label by the user, and third, the value u is an instance of the chosen class D.

■ **Binding rule #5 (textbox label filling).** Figure 4.19 shows the binding rule for an IOPE:TEXTBOX widget filled by a value of type string related to a IOPE:LABEL widget. Given the property p of the class C filled with value u of type string, the triggering of this binding rule will generate RDF triples to convey, first, the instance new_f has the value u via the property p, second, the human-readable label of value u is the entered string u_label by the user, and third, the value u is an instance of the class C.

■ **Binding rule #6 (textbox integer filling).** Figure 4.20 shows the binding rule for an IOPE:TEXTBOX widget filled by a value of type integer related to an IOPE:LABEL widget. Given the integer value k, triggering this rule will create RDF triples which connect k instances of the class C (i.e., $c_1 \ldots c_k$) to the instance new_f via the property p.

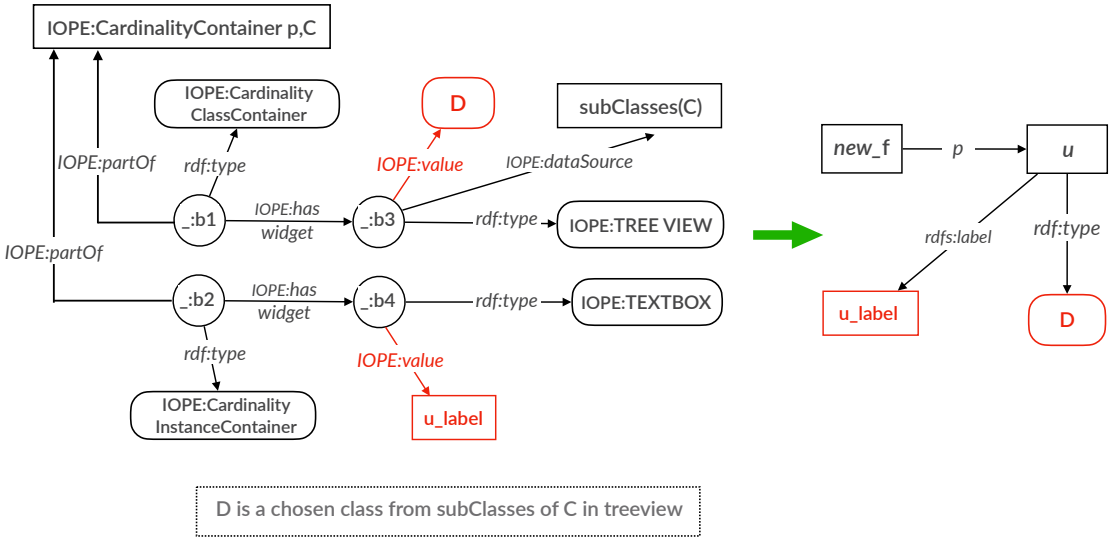

FIGURE 4.18: Binding rule #4, for a selected class D in an `IOPE:TREEVIEW` and its entered item(s) in an `IOPE:TEXTBOX` widget.

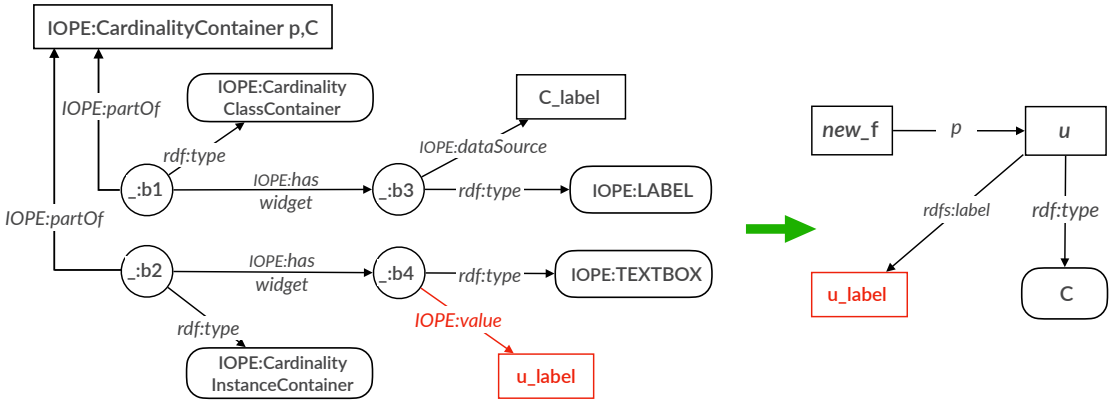

FIGURE 4.19: Binding rule #5, for the value of type string in an `IOPE:TEXTBOX` widget related to an `IOPE:LABEL` widget.

■ **Binding rule #7 (textbox treeview filling).** Figure 4.21 shows the binding rule for an `IOPE:TEXTBOX` widget filled by a value of type string related to an `Other_C` in an `IOPE:TREEVIEW` widget. For each chosen class `Other_C` (subclass of the class C) in `IOPE:TREEVIEW` widget, and the entered item(s) u in the `IOPE:TEXTBOX` widget, the binding rule generates RDF triples to convey, first, the instance new_f has the value u via property p, second, the human-readable label of value u is the entered string u_label by the user, third, the value u is an instance of the chosen class `Other_C`, and forth, the class `Other_C` is a sub-class of class C.

■ **Binding rule #8 (checkbox filling).** Figure 4.22 shows the binding rule for an `IOPE:CHECKBOX` widget filled by a value u_1 selected by the user. Once triggered, the rule will create an RDF triple which conveys that the instance new_f has the value u_1 via the property p.

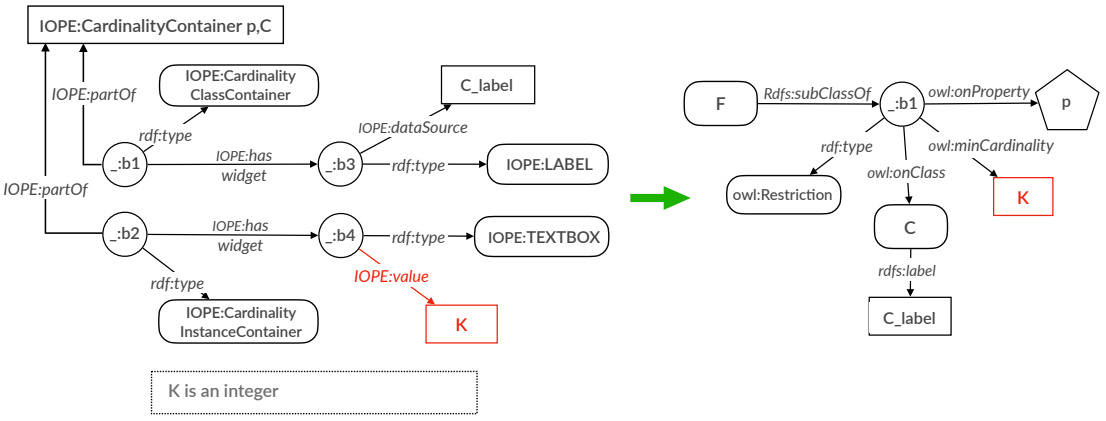

FIGURE 4.20: Binding rule #6, for a value of type integer in an `IOPE:TEXTBOX` widget related to an `IOPE:LABEL` widget.

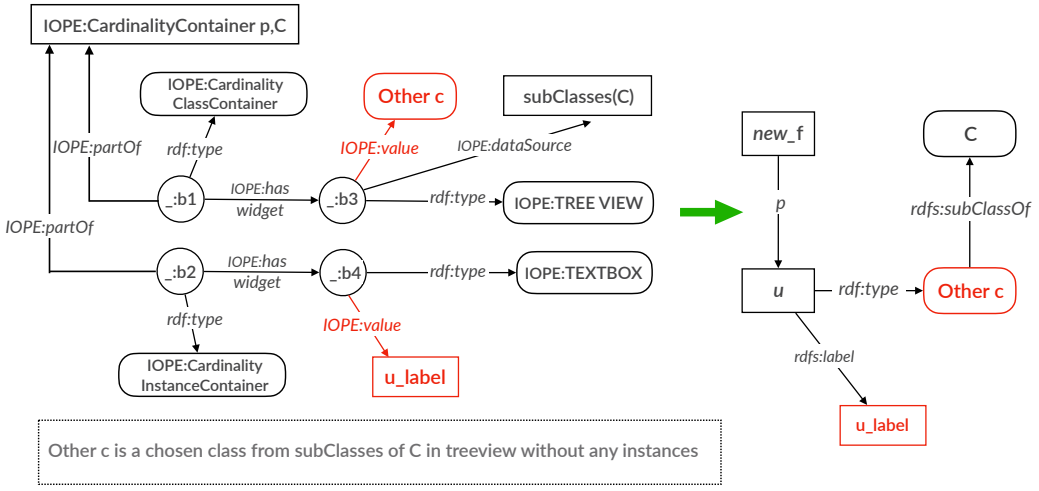

FIGURE 4.21: Binding rule #7, for an `IOPE:TEXTBOX` widget filled by a value of type string related to a class `Other_C`.

■ **Binding rule #9 (textbox integer filling where p is a datatype property.** Figure 4.23 shows the binding rule for an `IOPE:TEXTBOX` widget filled by a numerical value related to the datatype property p. Once triggered, the rule will create an RDF triple which conveys that the instance new_f has the numerical value k via the property p.

The set of all aforementioned binding rules contributes to the enrichment and population of the ontology. The resulting RDF graphs of the binding rules are first verified by ontology engineers, before being appended to the ontology.

To elaborate on the aforementioned binding rules, we proceed with our illustrative example from Section 4.3.2.2. Figure 4.24 left shows the resulting pre-filled Web page generated by the HTML implementation of the IOPEWEB specification. Some user interactions through the widgets are highlighted in this figure. Figure 4.24 right shows the transformation of those user interactions into RDF graphs as the outcome of triggering

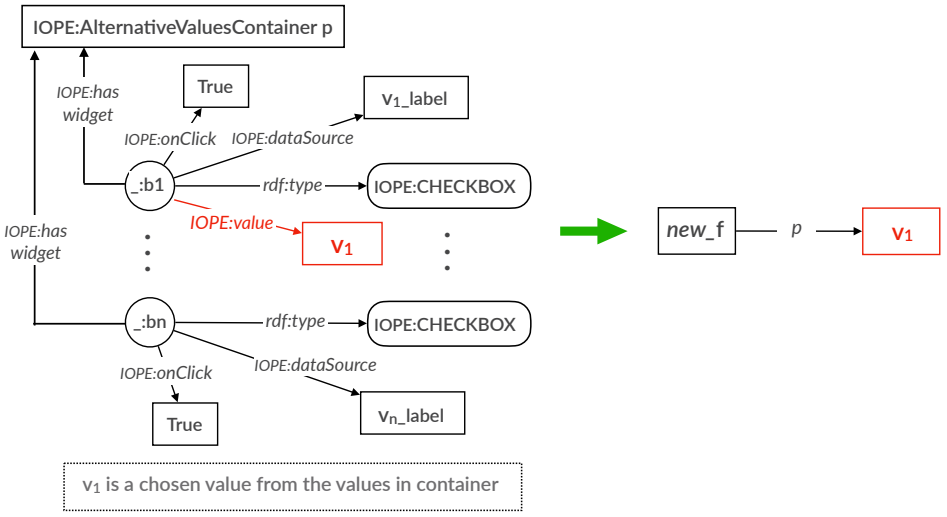

FIGURE 4.22: Binding rule #8, for an `IOPE:CHECKBOX` widget filled by selected value u_1.

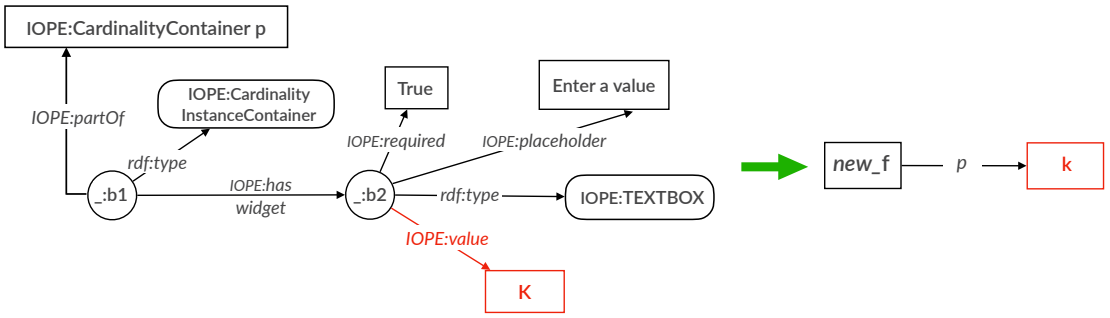

FIGURE 4.23: Binding rule #9, for an `IOPE:TEXTBOX` widget filled by a value of type integer related to the datatype property p.

the binding rules which are applicable to the widgets. We explain the four binding rules employed for transforming the interactions, i.e., the binding rules #1, #2, #3, and #6.

The binding rule #1 is triggered when the expert chose to create and explore the port-a-cath simulation training session (the first green arrow from the top). The generated RDF triple consists of a created instance `samsei:port_a_cath_placement_n3` of type `samsei:PortACathPlacement` class.

The binding rule #3 is triggered when two following user interactions are performed consecutively (the second green arrow from the top): (*i*) select the sub-class `samsei:DisposableDrape` from `samsei:ProtectiveSupplies` sub-class hierarchy, and (*ii*) select the `samsei:simple_disposable_drape` instance from the list of instances for the `samsei:DisposableDrape` class. Two RDF triples will be generated. The first

triple declares that the instance `samsei:port_a_cath_placement_n3` has the instance `samsei:simple_disposable_drape` via the property `samsei:equipmentSupplies`. The second one mentions that `samsei:simple_disposable_drape` is an instance of `samsei:DisposableDrape`.

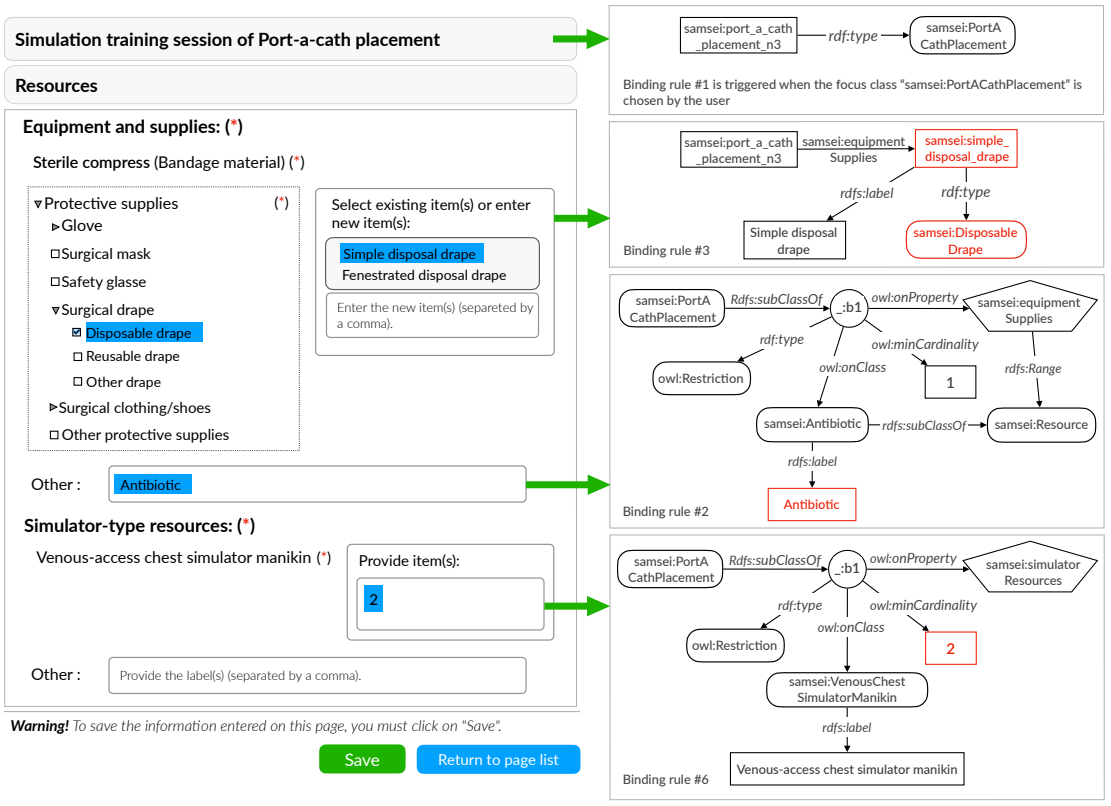

FIGURE 4.24: The IOPE resulting pre-filled Web page and the user interactions via the widgets (left), and the resulting RDF graphs by applying the binding rules (right).

In case the expert does not find the required resource, he/she is able to enter the name of the resource in the text box dedicated to a specific type of resource requested by the expert. The binding rule #2 is triggered when such new information (i.e., "Antibiotics" in our example) is added by the expert via `IOPE:TEXTBOX` widget in the free entry container of the property `samsei:equipmentSupplies` (the third green arrow from the top). The generated RDF triples express a new class `samsei:Antibiotic` and a new `cardinality` constraint for this simulation training session on the property `samsei:equipmentSupplies`. The new class `samsei:Antibiotic` is located as a sub-class of the class `samsei:Resource`, which is the range of property `samsei:equipmentSupplies`.

The binding rule #6 is triggered when the expert specifies the numerical value "2", conveying that his/her simulation training session requires two venous-access chest simulator manikins (the first green arrow from the bottom). The generated RDF triples connect 2 instances of the class `samsei:VenousChestSimulatorManikin` to the instance

`samsei:port_a_cath_placement_n3` via the property `samsei:simulatorResources`.

4.4 Summary

In this chapter, we presented the second contribution of this thesis, i.e., a framework called IOPE for the automatic construction of a Graphical User Interface (GUI) consisting of pre-filled Web pages. This contribution is in the sequel with the contributions presented in Chapter 3, where the constructed GUI operates on top of the ontology enhancement layer to complete the loop of the ontology engineering approach and enable fruitful interactions with the user for ontology updates. We mentioned how the literature addresses data publishing in ontology updates, by reviewing ontology editing tools, as well as graph-based and form-based approaches. Two common challenges among the related work are the followings: (*i*) lack of usability and intuitiveness of the updating interfaces, and (*ii*) high complexity where experts with no the prior knowledge of the formal syntax and the semantics of ontology languages are often left alone.

We presented the core idea behind our approach as "transposing the RDF data and the ontological constraints of a given domain ontology into a GUI" to enable an interactive data publishing approach for ontology updates. The resulting GUI functions as a guidance for domain experts to easily explore the ontology and update it through interactive graphical widgets. The resulting GUI is a set of web pages that are automatically generated and pre-filled in a declarative approach based on a set of mapping rules, which map RDF constraint graphs to Web form templates. The Web form templates are described using a Web form ontology called IOPEWeb. We presented different components of the IOPEWeb ontology to model widgets, containers, and page layouts. We then provided the different mapping rules to map components of each ontological constraint to the widgets inside the containers modeled by IOPEWeb.

The input entered through user interactions in the GUI are then be bound to RDF data to update the ontology. We presented the different binding rules whose role is to specify how to transform user interactions into RDF graphs. The resulting RDF graphs of the binding rules are first verified by ontology engineers, and then will be added to the ontology, in the form of ontology enrichment and population.

Chapter 5

Evaluation

5.1 Introduction

In this chapter, we evaluate the contributions of this thesis on ontology construction (detailed in Chapter 3) and ontology update (detailed in Chapter 4). We consider the three following objectives for our evaluation:

- First, we measure the usefulness of IOPE interface in enabling experts to populate and enrich the OntoSAMSEI ontology;
- Second, we measure the quality of the OntoSAMSEI ontology through its presentation by the IOPE GUI;
- Third, we shed light on the generality aspects of our approach, and verify whether IOPE can be easily adapted to ontologies in other domains as well.

This chapter is organized as follows. In Section 5.2, we present the evaluation methodology that we employed to measure the quality of our framework through an in-depth expert study. Then we discuss the results of our study for the first evaluation objective (i.e., evaluating IOPE) in Section 5.3. Next, we present the experts' viewpoints on the quality of OntoSAMSEI (i.e., the second evaluation objective) in Section 5.4. Section 5.5 covers the third evaluation objective, where we adapt IOPE to an ontology in another domain, and discuss its applicability. Last, we summarize this chapter in Section 5.6.

5.2 Evaluation Settings

Given the interactive nature of our framework, *user studies* are an indispensable part of our evaluation protocol, where the utility of the system is not necessarily measured based on typical objective measures, but based on the user feedback. However, as we deal with specialized ontologies in this thesis, it is infeasible to deploy our user study campaign over crowdsourcing marketplaces such as Amazon Mechanical Turk[1] and Prolific[2], whose participants are average information consumers [Ipe10], and not necessarily the experts of the given specialized ontology. This is why we designed our own protocol for an in-depth *expert study*. While an expert study has often fewer participants than large-scale user studies (because experts are scarce and unavailable), the former gains richer and deeper insights by employing participants who are highly knowledgeable about the domain under investigation.

For our study, we employed a group of 22 medical experts in the domain of simulation-based training in Medicine. These experts are a subset of the population we solicited one year prior to the expert study, to perform the bootstrapping for the OntoSAMSEI ontology (see Section 3.3.1). The general assumption about these experts is that they have a deep understanding of their own domain, but lack familiarity with RDF and OWL. The hypothesis is that IOPE helps these experts interact with their ontology, i.e., OntoSAMSEI, in an intuitive and user-friendly fashion, with the objective of enriching and populating the ontology.

The protocol of our expert study consists of two consecutive steps: interaction step and evaluation step.

Interaction step. In this first step, the expert logs into IOPE with the credentials communicated to him/her before the study, selects a simulation training session of interest, and begins to observe and update the information presented in the pre-filled Web pages. In this step, not only we record the interactions, we also keep track of some meta-data, such as the total time spent on updating a training session, and the total number of interactions. Once the expert is done with updating the Web pages of one training session, he/she is free to come back to the list of all training sessions, and choose another session to observe and update. Alternatively, the expert can terminate the update process and move to the second step of the study.

Evaluation step. In the second step, the expert responds to a survey which evaluates some qualitative aspects of IOPE and its underlying ontology (i.e., OntoSAMSEI).

[1] *https://www.mturk.com*
[2] *https://www.prolific.co*

The survey captures the expert's viewpoint regarding his/her interaction experience with the pre-filled Web pages.

Our user study data contains interaction logs (from the interaction step) and survey responses (from the evaluation step). For a more granular analysis of the results, we propose to group the experts based on the intensity and duration of their activity. Note that to respect the privacy of the experts, we build groups based on the behavior and not the identity of the participants, where "expert behavior" is captured by the variables *number of interactions* and *interaction duration*.

Expert groups based on the number of interactions. We define the groups of *prolific*, *active*, and *moderate* experts. Members of the first group performed more than 6 interactions with IOPE, while the number of interactions is between 3 and 6 for the second group, and less than 3 interactions for the third group. Table 5.1 shows the distribution of experts in these groups.

Table 5.1: Distribution of experts in interaction number groups.

	Moderate experts	**Active experts**	**Prolific experts**
Expert population	22.73%	50%	27.27%

Expert groups based on the interaction duration. We define groups of *short-time*, *medium-time*, and *long-time* experts. The average interaction duration for the members of these groups is less than 2 minutes, between 2 and 4 minutes, and more than 4 minutes, respectively. Table 5.2 shows the distribution of experts in these groups.

Table 5.2: Distribution of experts in interaction duration groups.

	Short-time experts	**Medium-time experts**	**Long-time experts**
Expert population	50%	31.82%	18.18%

Table 5.3 reports the distribution of the interaction duration groups for each interaction number group, where each column sums up to 1.0. We observe that more interactions do not necessary yield to more time spent to interact. This shows that IOPE helps experts fulfill their task in a reasonable amount of time, even for prolific experts.

Table 5.3: Distribution of interaction duration groups in interaction number groups.

		Interaction number groups		
		Moderate experts	**Active experts**	**Prolific experts**
Interaction duration groups	**Short-time experts**	0.80	0.46	0.33
	Medium-time experts	0.00	0.27	0.67
	Long-time experts	0.20	0.27	0.00

In Sections 5.3 and 5.4, we elaborate on the results of our expert study for IOPE and OntoSAMSEI, respectively, and compare different expert groups together.

5.3 Evaluation of IOPE Interface

Our aim in this section is to evaluate different aspects of IOPE by providing an answer for the following questions regarding the efficiency and effectiveness of the IOPE interface.

- **Q1: Engagement.** Do the experts engage with the IOPE interface and the interactions therein?
- **Q2: Time-to-insight.** How does IOPE impact the "time-to-insight" for the experts?
- **Q3: Added value.** What is the added value of IOPE compared to a standard ontology editor?
- **Q4: Satisfaction.** Are the experts satisfied with the IOPE interface for observing and updating their specialized ontology?

Sections 5.3.1 to 5.3.4 provide answers for the above questions.

5.3.1 IOPE's Expert Engagement

To elaborate on **Q1** (engagement), we provide a summary of the experts' experience in our study, as a proof of concept. By aggregating the collected meta-data for the 22 experts, we observe that the average time that an expert spent on updating training sessions using the IOPE interface is 163 seconds (i.e., 2.72 minutes). After detecting and removing outlier data (i.e., meta-data which correspond to experts with absolute zero spent time, or with an extremely long spent time due to quiescence), we recorded 320 seconds (i.e., 5.33 minutes) as the maximum time spent on updating, and 67 seconds (i.e., 1.12 minutes) as the minimum. Moreover, the experts performed 5.78 interactions with IOPE interface, on average. We also recorded a maximum of 14 and a minimum 3 interactions. The majority of interactions are with the `IOPE:CHECKBOX` widget (i.e., 56.15% of the interactions) followed by `IOPE:TEXTBOX` widget (32.30% of the interactions) and `IOPE:LISTBOX` widget (11.53% of interactions). Compared to pilot studies and in-person interactions with the experts, these results collectively show a reasonable engagement of the experts with the IOPE platform and its interaction model.

5.3.2 IOPE's Time-to-Insight

Next, we focus on **Q2**. "Time-to-insight" is a typical evaluation criteria in interactive systems [RJN20] which measures the expected amount of time (in seconds) for the experts to fulfil their task. The notion of "task" here refers to ontology updating. Given the interactions with IOPE as a training step, we asked the experts, in the evaluation part of our study, about their prediction on the estimated time-to-insight for a future utilization of the system: *"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 the choice "1" means "very short time", and the choice "5" means "very long time".

Figure 5.1 shows the results. We observe that on average the majority of experts chose either "short time" or "average time", i.e., options 2 and 3 in the Likert scale. Moreover, it turns out that the prolific experts and long-time experts perceive shorter expected time compared to the active and moderate experts. A possible interpretation is that more interactions and more time spent interacting with the system boosts the perception of faster delivery of required information.

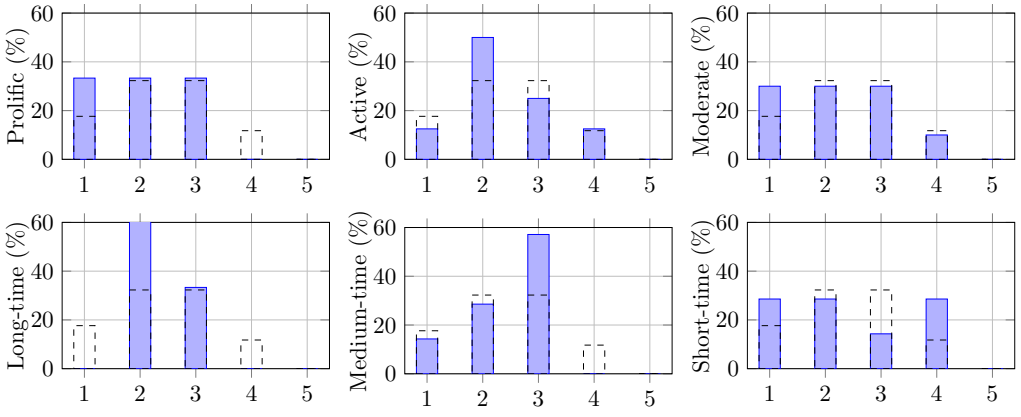

FIGURE 5.1: Prediction of experts about time-to-insight for their next utilization of the IOPE interface. Dashed bars show the average values of time-to-insights for all the experts.

5.3.3 IOPE's Added Value

To elaborate on **Q3** (added value), we compare IOPE with a standard state-of-the-art ontology editor, i.e., TOPBRAID [PCHK20]. The comparison is built on the test bed illustrated in Table 5.4, i.e., we measure the number of interactions required to fulfill ontology editing tasks. The tasks are categorized into three levels of difficulty, based on the "cognitive bias" task taxonomy proposed in [DFP+20]. Given the assumption about the experts' competence discussed in Section 5.2 (i.e., experts do not necessarily have

TABLE 5.4: Testbed for the comparison between IOPE and TOPBRAID.

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

the basic knowledge to work with ontology editing tools), this comparative study was performed by 5 collaborators of this thesis, who have the knowledge of both the domain and the tools (IOPE and TOPBRAID).

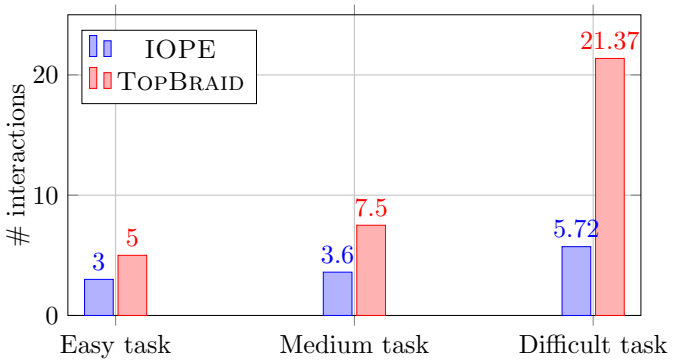

FIGURE 5.2: Average number of interactions in IOPE and TOPBRAID.

Figure 5.2 shows the results of the comparison 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 steps for an easy task to average 5.72 steps for a difficult task, while using TOPBRAID grows from average 5 steps for an easy task to average 21 steps for a difficult task. This shows that IOPE enables the experts to fulfill their tasks more rapidly by weaving relevant information together using ontological constraints. 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.

TABLE 5.5: User satisfaction aspects.

Measures	**Definition**	**Question asked in the survey**
Utility [Tho05, AT13]	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 [RJN20, AT13]	The easiness of interactions with the method	To which degree do you find IOPE easy-to-use?
Adoption [Tho05]	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?

5.3.4 IOPE's Expert Satisfaction

To elaborate on **Q4** (satisfaction), we define the notion of *user satisfaction* as a combination of "utility", "usability", and "adoption" [Tho05]. Table 5.5 provides a definition for these user satisfaction aspects. In the evaluation part of our expert study, we measure these aspects on a Likert scale from 1 to 5, which captures the assessment of experts on their satisfaction when interacting with IOPE. For instance, as assessment of 4 for "usability" means that the expert finds IOPE useful for most ontology update tasks. The last column in Table 5.5 mentions the questions asked from the experts in the survey.

The results are shown in Figure 5.3. We observe that 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 smooth the learning curve and increases the perception of utility, which is also confirmed by long-time experts who are entirely on the positive spectrum.

Also, 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 fulfill their tasks and eventually got lost in the process.

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. The semantics of adoption is perpendicular to the interaction volume and interaction time, hence no obvious correlation was observed in expert groups.

Table 5.6: OntoSAMSEI evaluation measures.

Measures	Definition	Question asked in the survey
accuracy [OA19, RJN20]	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 [RJN20]	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?

5.4 OntoSAMSEI Evaluation

The experts in our study interacted with the IOPE interface to ultimately perform ontology updates on the domain ontology. OntoSAMSEI is the domain ontology that

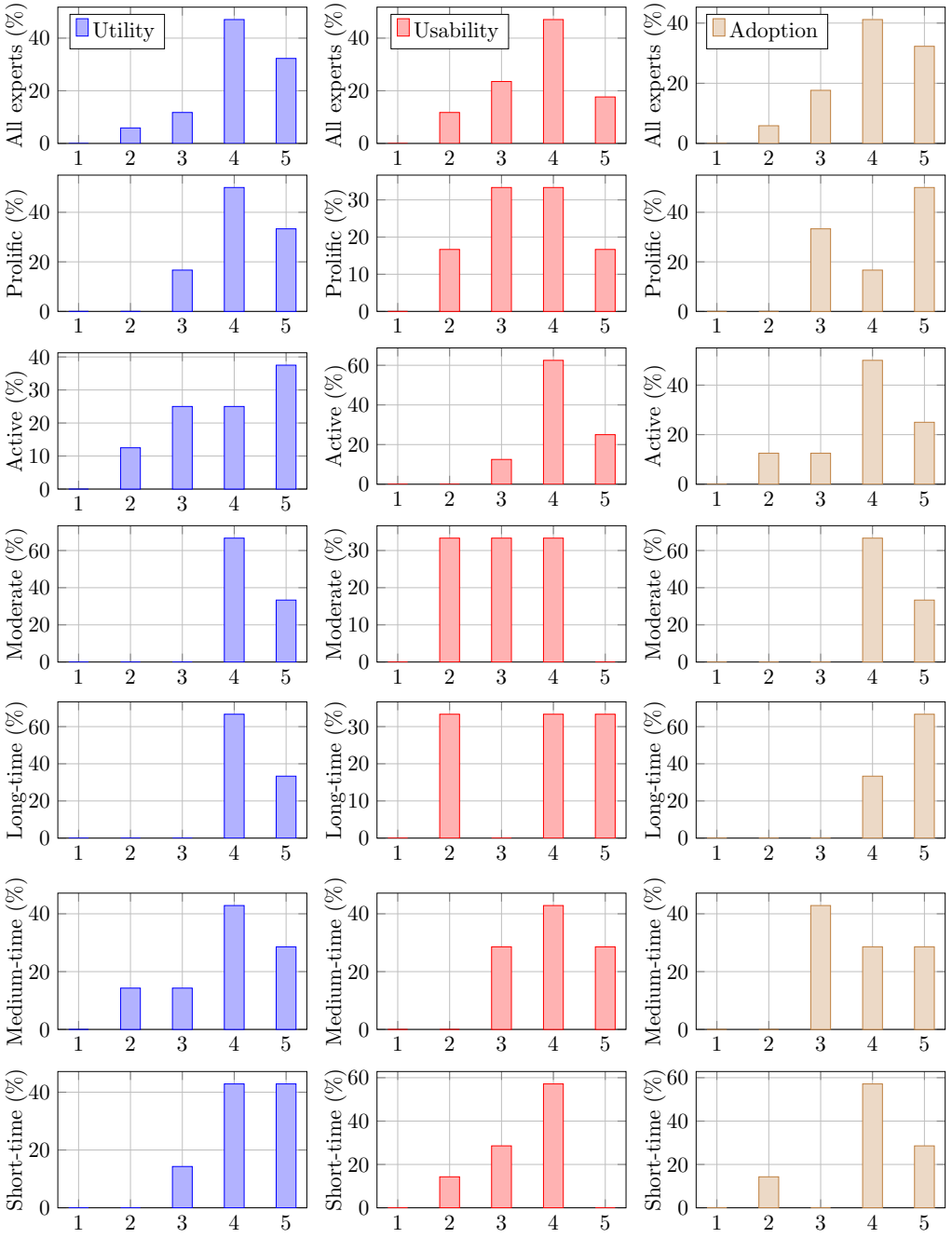

FIGURE 5.3: Experts' assessment on satisfaction aspects.

we developed in Chapter 3 of this thesis as the outcome of our 4-step ontology engineering process. In this part of the experiment, our goal is to evaluate the ONTOSAMSEI ontology itself.

We measure the experts' assessment of *accuracy* and *completeness* for the ONTOSAMSEI ontology through its presentation to the experts by IOPE GUI. Table 5.6 defines these measures.

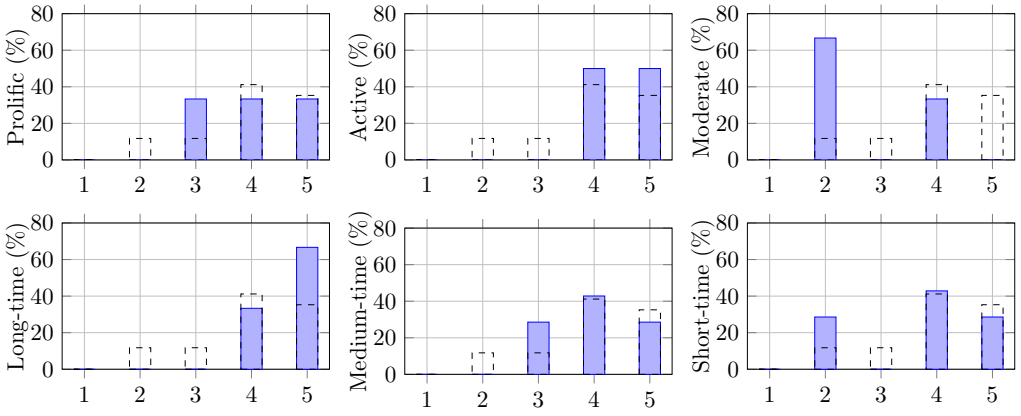

FIGURE 5.4: Experts' assessment of ONTOSAMSEI's accuracy. Dashed bars show the average accuracy for all the experts.

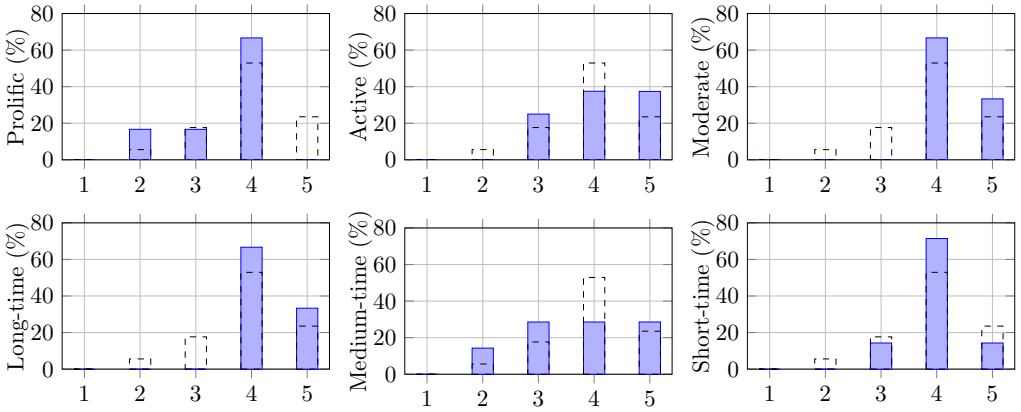

FIGURE 5.5: Experts' assessment of ONTOSAMSEI's completeness. Dashed bars show the average scores of completeness for all the experts.

The last column in Table 5.6 mentions the questions that we asked in the evaluation part of our expert study. For both accuracy and completeness, we receive the expert feedback using a Likert scale from 1 to 5, where 5 means high accuracy and high completeness, respectively.

Figures 5.4 and 5.5 show the results. We observe in Figure 5.4 that 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. Regarding completeness, we observe in Figure 5.5 that 76.46% of the participants find ONTOSAMSEI "adequately complete". 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.

5.5 Generality of IOPE

Sections 5.3 and 5.4 present evaluations of IOPE when OntoSAMSEI is employed as the underlying ontology. In this section, our goal is to shed light on the generality aspects of IOPE by showing its applicability on another domain. Given an ontology from another domain, we verify whether IOPE is able to automatically generate pre-filled Web pages which are meaningful in that domain.

We employ an ontology called PerSCiDO which contains the structure and information of a dataset publishing and sharing web portal[3]. While the portal has the same functionality as Harvard's Dataverse[4] and Google Dataset Search[5], it is among the few services that enables experts to employ SPARQL queries to explore the datasets.[6] The ontology is organized around a main class `perscido:Dataset` with properties describing the datasets, such as title, description, scientific fields, and keywords. The two main differences between OntoSAMSEI and PerSCiDO are as follows: (*i*) there exist several focus classes in the former, while there exist a single one in the latter (i.e., the dataset), (*ii*) The PerSCiDO ontology is entirely described with "domain and range" ontological constraints. However, some properties are indicated as mandatory in the interface of PerSCiDO, in which it is hard-coded by a red asterisk. For these properties, we have added the ontological constraints "minCardinality 1" in the PerSCiDO ontology.

We apply IOPE to the "dataset submission" functionality of PerSCiDO. A series of HTML forms are manually crafted in the web portal, where the expert enters the desired values in the widgets for submitting a new dataset, and the values will be fed to the ontology upon clicking the "validate" button. Figures 5.6 and 5.7 show two example forms of the data submission process, for entering information about "content description" and "data processing", respectively. The left side of the figures show the pre-filled Web forms automatically generated by IOPE, and the right side illustrates the same forms, manually crafted in the web portal.

In Figure 5.6, we observe that all the elements in the manual form are well presented in the IOPE's output. Moreover, the dynamic structure of the auto-generated forms facilitates changes in the future, while the manual form is fixed and needs to be re-engineered. For example, a new "scientific field" can be easily integrated in the IOPE's generation process, while it requires tedious manual work in the status quo.

[3] *https://perscido.univ-grenoble-alpes.fr*
[4] *https://dataverse.harvard.edu*
[5] *https://datasetsearch.research.google.com*
[6] *https://perscido.univ-grenoble-alpes.fr/sparql_query*

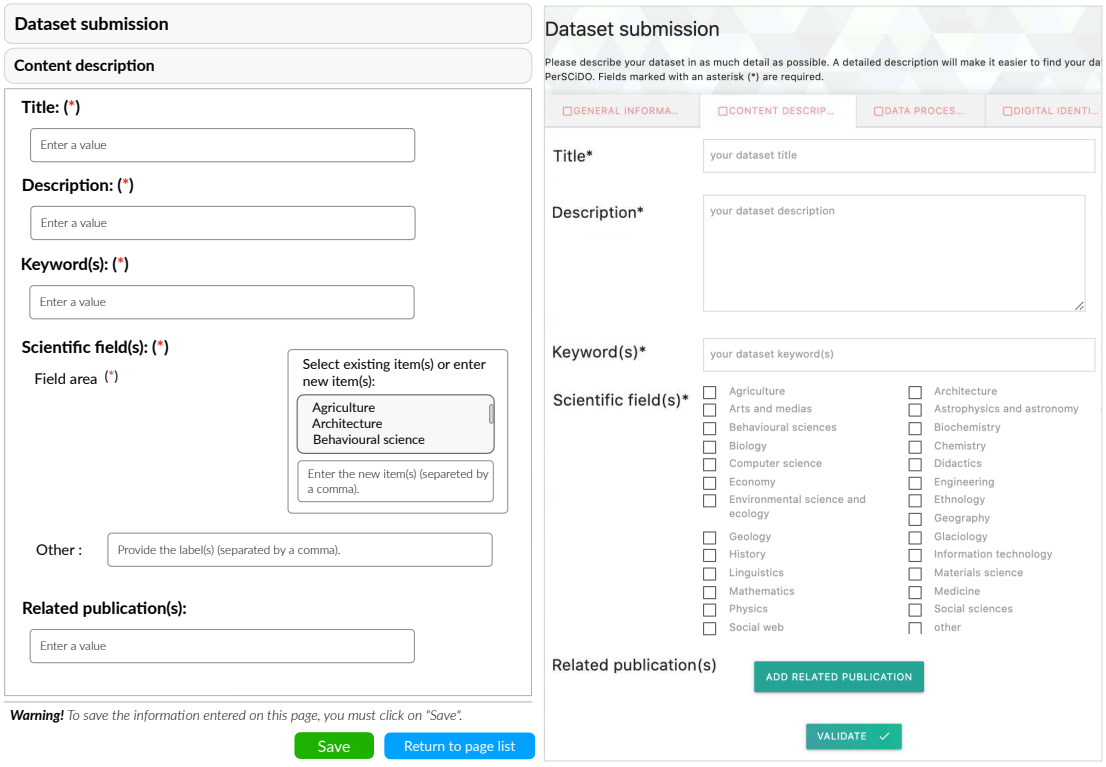

FIGURE 5.6: Content description step for submitting a new dataset in PERSCIDO.

In Figure 5.7, beyond the complete presentation of the entire manual form, IOPE renders a more homogeneous output, as the same rules are applied for the same type. For instance, the manual form represents the options of "data type" as a list, and the options of "automatic tasks" as check boxes. While this inconsistency originates from typical arbitrarily decisions of front-end engineers, auto-generated forms in IOPE ensures the maximal consistency, and hence provide more intuitiveness and easy-friendliness to experts, for updating their ontologies.

5.6 Summary

In this chapter, we reported the results of our experiments on ontology construction and ontology update. Given the interactive nature of our framework, we built an expert study where the participants interact with IOPE's automatically generated Web forms and respond to a survey about their experience. We showed that the experts are engaged to the Web forms, as they need a short time-to-insight to update their ontology in IOPE. We also discussed the added value of our approach in comparison with a state-of-the-art ontology editing tool, in decreasing the number of interactions to fulfill the task.

In terms of user satisfaction, we mentioned that high values of utility, usability, and adoption are perceived for our framework. We also illustrated the positive assessment

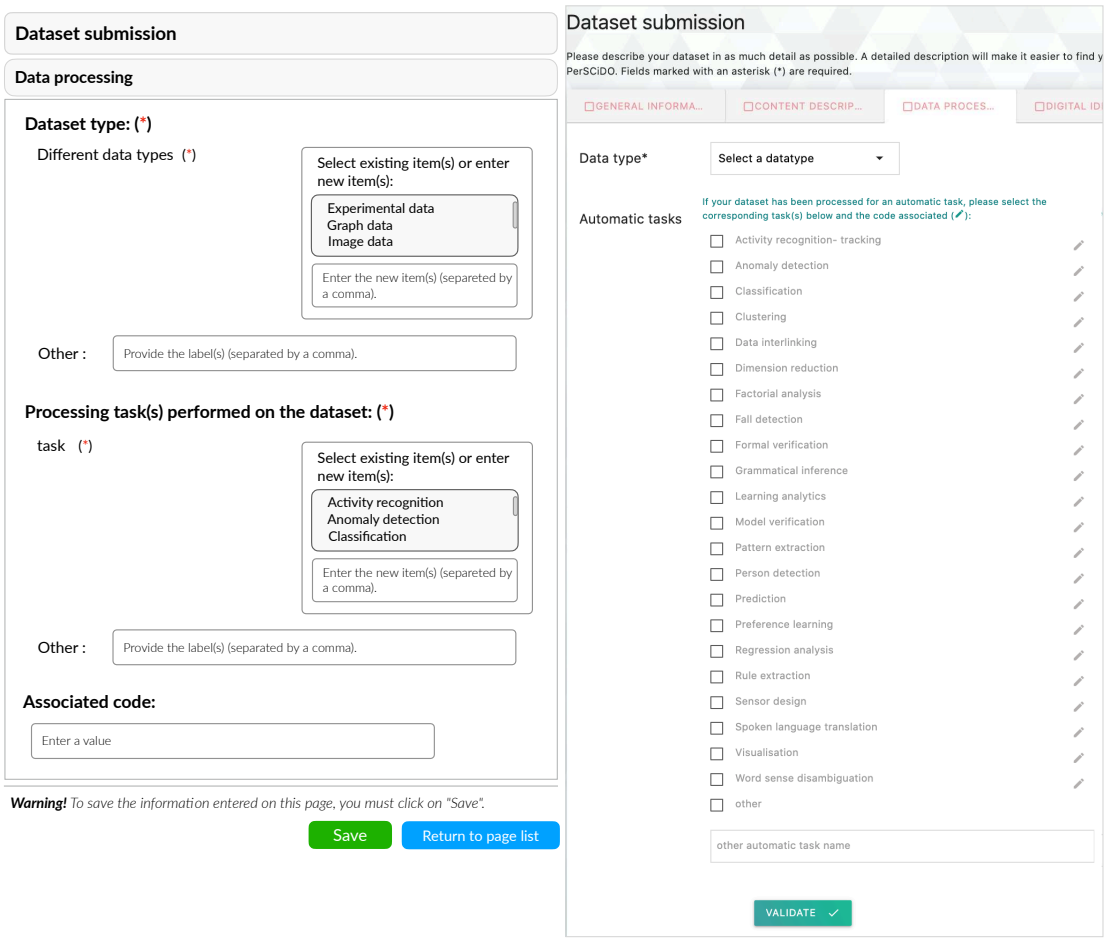

FIGURE 5.7: Data processing step for submitting a new dataset in PERSCIDO.

of experts on the accuracy and completeness of the ONTOSAMSEI ontology, when presented by IOPE GUI.

Last, we shed light on the generality aspects of our approach and showed how IOPE contributes to domains other than Medicine by providing more dynamicity and consistency.

Chapter 6

Summary and Perspectives

6.1 Summary

The focus of this thesis is on *constructing specialized ontologies* to capture the skills of experienced experts in a particular domain (i.e., simulation-based medical education), with the goal of sharing those skills with a larger community of trainees and less experienced experts in the domain. To achieve this objective, we followed two distinct directions: (*i*) knowledge elicitation and formalization of the simulation-based medical education, and (*ii*) development of an interactive ontology update approach. The combination of the two directions constitute the full pipeline of ontology engineering for specialized domains, from ontology construction to ontology enrichment and population (i.e., ontology update).

Chapters 1 and 2 of this thesis presented the introductory and preliminary concepts used in our contributions to realize the two above directions. We discussed the challenges of ontology construction for ill-defined domains, and mentioned pedagogical domains as a mainstream, where simulation-based medical training is an example. We also provided formal definitions of some fundamental notions in this thesis, such as semantic Web, semantic Web languages, and ontological constraints.

In Chapter 3, we presented the first contribution of this thesis, i.e., a methodology for designing and engineering a simulation-based medical training ontology, called OntoSAMSEI. We discussed the challenges associated to this objective, i.e., the scarcity of formal models and documentations in ill-defined domains including simulation-based medical education. We discussed three groups of ontology engineering methods (OEMs), i.e., non-collaborative, collaborative, and custom OEMs, and concluded that collaborative OEMS are more expressive and powerful for building ill-defined domains. We

presented our 4-step collaborative OEM, which begins by ontology bootstrapping, followed by knowledge elicitation and enhancement. The resulting ontology is a hierarchy of classes and of properties, enriched by ontological constraints on the properties and on the classes. Last, we presented the fourth step of our approach, i.e., interactive ontology update, to account for the evolving nature of ontologies.

In Chapter 4, we presented the second contribution of this thesis, i.e., a framework called IOPE for the automatic construction of a Graphical User Interface (GUI) consisting of pre-filled Web pages. IOPE controls the update process of the input ontology, and hence completes the engineering pipeline of Chapter 3. We discussed the challenges associated to automatic GUI construction for ontologies, i.e., the difficulty of graphically presenting the graph-based nature of ontologies, and making those GUIs intuitive and easy-to-use for the experts without the knowledge of the formal syntax and semantics of ontology languages. We mentioned how the literature addresses ontology updates, by reviewing ontology editing tools, as well as graph-based and form-based approaches. We presented the main idea behind IOPE as transposing the RDF data and the ontological constraints into a GUI, using mapping rules. These automatically generated GUIs provide guidance for domain experts and facilitate the ontology exploration and update through interactive graphical widgets. We also discussed binding rules, to bind the input entered through user interactions in the GUI to RDF data, and consequently perform ontology updates.

In Chapter 5, we evaluated our contributions on ontology construction and ontology update. We defined three distinct objectives for our evaluation scheme: (*i*) the benefit of the IOPE interface for enabling experts to populate and enrich the OntoSAMSEI ontology, (*ii*) the quality of the OntoSAMSEI ontology through its presentation by the IOPE GUI, and (*iii*) the extent of IOPE's generality in domains other than Medicine. Given the interactive nature of our framework, we motivated the construction of an in-depth expert study, where the participants interact with IOPE's automatically generated Web forms and respond to a survey about their experience. For the first direction of our evaluation, we measured different aspects of IOPE functionality using the survey responses, such as engagement, time-to-insight, added value, and expert satisfaction. We showed that the experts are engaged to the Web forms, as they need a short time-to-insight to update their ontology. We also mentioned that high values of utility, usability, and adoption are perceived for our framework, hence high expert satisfaction. For the second direction, we illustrated the assessment of experts on the accuracy and completeness of the OntoSAMSEI ontology, when presented by IOPE GUI. For the third direction, we depicted the applicability of our approach to domains other than Medicine, by adapting IOPE to PerSciDO, a dataset sharing and publishing web portal.

6.2 Perspectives

Among several future directions that we envision we focus on the following three perspectives.

Direction 1: Learning mapping rules. We discussed in Section 5.5 that IOPE can be applied to ontologies from different domains. These domains can be described by different ontological constraints such as "cardinality", "hasValue", and "domain and range". However, it may be the case that an ontology is described only with "domain and range" ontological constraints. PerSCiDO is an example of such domains, where the ontology is entirely constructed using RDFS ontological constraints. In such cases, it is beneficial to devise an automatic way to extend the list of mapping rules using the resources which are already available for the domain under investigation, such as Web pages and Intelligent Tutoring Systems (ITS) [PMMB88, SKS+16]. Our first direction of future work is to employ supervised learning methods to automatically learn a set of mapping rules from the ontology and the resources of a given domain.

To learn mapping rules, the learning method receives as input different RDF triples of the ontology paired with their corresponding resources, and learns the relation between those triples and the resources, hence the mapping rules. To decrease the learning bias [NFG+20], the resources should be collected from diverse sources. For instance, in the case of the PerSCiDO ontology, one can obtain Web pages from different services following the same goal, e.g., Dataverse and Google Dataset Search. Also in the case of the OntoSAMSEI ontology, various ITSs such as CIRCSIM-Tutor [EM06] and Cardiac-Tutor [EW95] can be employed. The learned mapping rules will be able to generate optimal Web pages (i.e., ones which exactly capture the content of the ontology) and expand the list of mappings beyond "domain and range" constraints.

Direction 2: Explainability. In the Artificial Intelligence (AI) community, there exist a growing demand for explaining models which assist users in the process of decision making [NJ17, AB18], which is often called XAI. IOPE is an AI system whose guidance is rendered in a rule-based fashion. An advantage of rule-based systems is that they are transparent-by-design, hence naturally explainable. For instance, if an expert wonders why a Web page is generated in a particular form, IOPE can easily reveal the mapping rule responsible for that generation, as an explanation. The XAI in IOPE can be improved by providing on-screen and widget-level hints for the experts who are less familiar with the technicality of the widgets.

Beyond the natural transparency in IOPE, we observed in our expert study (Section 5.3) that some domain experts experienced difficulties in interacting with the generated Web pages, due to their lack of knowledge about the technicality of the widgets. For instance,

they might not be aware a-priori that more than one option can be selected in a list, or no characters should be entered in a text box whose placeholder is "enter a number". This is a classical problem in the HCI domain, where *widget hints* are proposed to explain to the user the way that he/she should employ that widget. A seminal work is Scented Widgets [WHA07] where each widget is enhanced with embedded visualizations to facilitate the exploration in the Web pages. Our second future direction is to integrate widget hints into the IOPEWeb ontology and enable widget-level explanations. This enhanced XAI feature will increase the experts' engagement to the platform and prevents them from jump shipping and abandoning the interface.

In IOPE, widget-level explanations should be based on both the ontology and the mapping rules. IOPE interface can be enriched with clickable question mark icons next to each widget, which the expert can refer to, if needed. A natural form of explainability is a "question answer" pair [ZCA+18], where clicking a question mark icon reveals such pair to clarify the interaction. A set of question templates can be prepared (e.g., "intelligibility type questions" [ZMA19]) whose answers will be dynamically assigned based on the content of the ontology and the mapping rules.

Direction 3: Automatic validation of ontology updates. In our setting, each ontology update should be first validated by an ontology engineer before being permanently added to the ontology. This is an essential step in the context of engineering a specialized ontology such as OntoSAMSEI, as the information are sensitive and the penetration of erroneous and/or redundant entries is strictly forbidden. In more general ontologies, however, the process can be more robust by leveraging *group consensus methods* [AORS15] (majority voting, least misery, pairwise disagreement, etc.) among the experts interacting with the IOPE interface. Our third future direction is to automate the validation process for general ontologies by leveraging *the wisdom of the crowd*, i.e., the experts are scrutinized about the updates that other experts had already performed, and if the consensus on an update reached some certain level, it can be automatically added to the ontology, without the need of an approval from an ontology engineer. Given a threshold of experts vote to approve an ontology update, it will be ultimately added to the ontology.

As the ontology updates can be abundant, an auto-validation method should be able to select a subset of those updates to present to the expert. The strategy to query such subset of the ontology updates should be based on maximizing the validation votes (both approvals and rejections). In other words, there is more necessity in voting for validating an ontology update which has never been validated before, compared to another one which has already received some votes. Hence the updates with fewer validation votes should have more chances to be queried. In long term, such query strategy ensures

that almost all ontology updates get enough validation votes. This query strategy can be modeled in the framework of Active Learning (AL) [DPD16, Set09], whose main objective is to find the best set of unlabeled data to be labeled by data annotators, using a query strategy. In AL, the strategies are typically entropy-based to ensure that the selected subset has the largest impact on label completion. Our future work is inspired by these efforts to come up with query strategies in the context of ontology updates.

Appendix A

Online Questionnaire

In Chapter 3, we presented a 4-staged collaborative ontology engineering approach for the specialized domain of simulation-based medical training. The four consecutive steps of the process are *ontology bootstrapping*, *knowledge elicitation*, *enhancement*, and *update*. Once the initial ontology is bootstrapped, we disseminate an online questionnaire among the domain experts to elicit the domain knowledge.

As instructed in Section 3.3.2, we teamed up with a pedagogical engineer to design the online questionnaire with 7 consecutive sections to be filled by the experts. These steps will collectively guide the expert in building a new simulation-based training session. The steps of the questionnaire are as follows: general description (Figure A.1), target audience (Figure A.2), goals (Figure A.3), prerequisites (Figure A.4), resources (Figure A.5), conditions and risks (Figure A.6), and additional information (Figure A.7).

In each step of the questionnaire, the expert can save the training session until that point, and complete the remaining parts later. It is also possible for the expert to come back to previous steps and change already entered information.

Moreover, to address the privacy concerns of the experts and respect the GDPR context[1], we enable the experts to have full access to their data, and remove the entries if they desire. We also guarantee that the collected data is only used in the context of this thesis.

[1] *General Data Protection Regulation (GDPR): https://www.gdpr-info.eu*

Figure A.1 shows the first step of the online questionnaire. This step deals with the aspects of data lineage and data ownership, where the information about the health educator (i.e., the session author) is captured. As discussed in Section 4.3.2.2, a red asterisk in front of a widget shows that the widget is mandatory and must be filled by the expert.

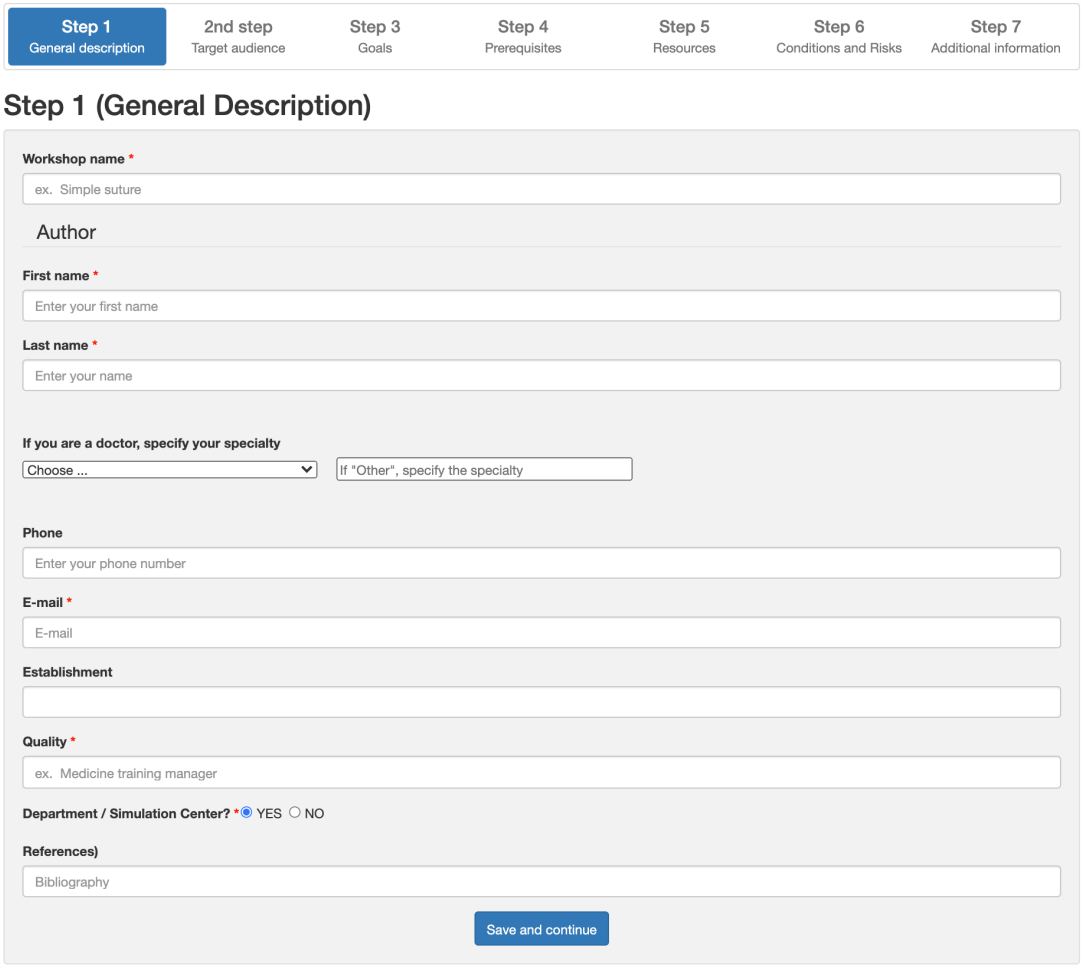

FIGURE A.1: Step 1 in the online questionnaire (general description)

Figure A.2 shows the second step of the online questionnaire. In this step, the educator defines the audience that the training session should target. Most options in this step are based on the French education system in the domain of Medicine [SJH+07, PJN+19].

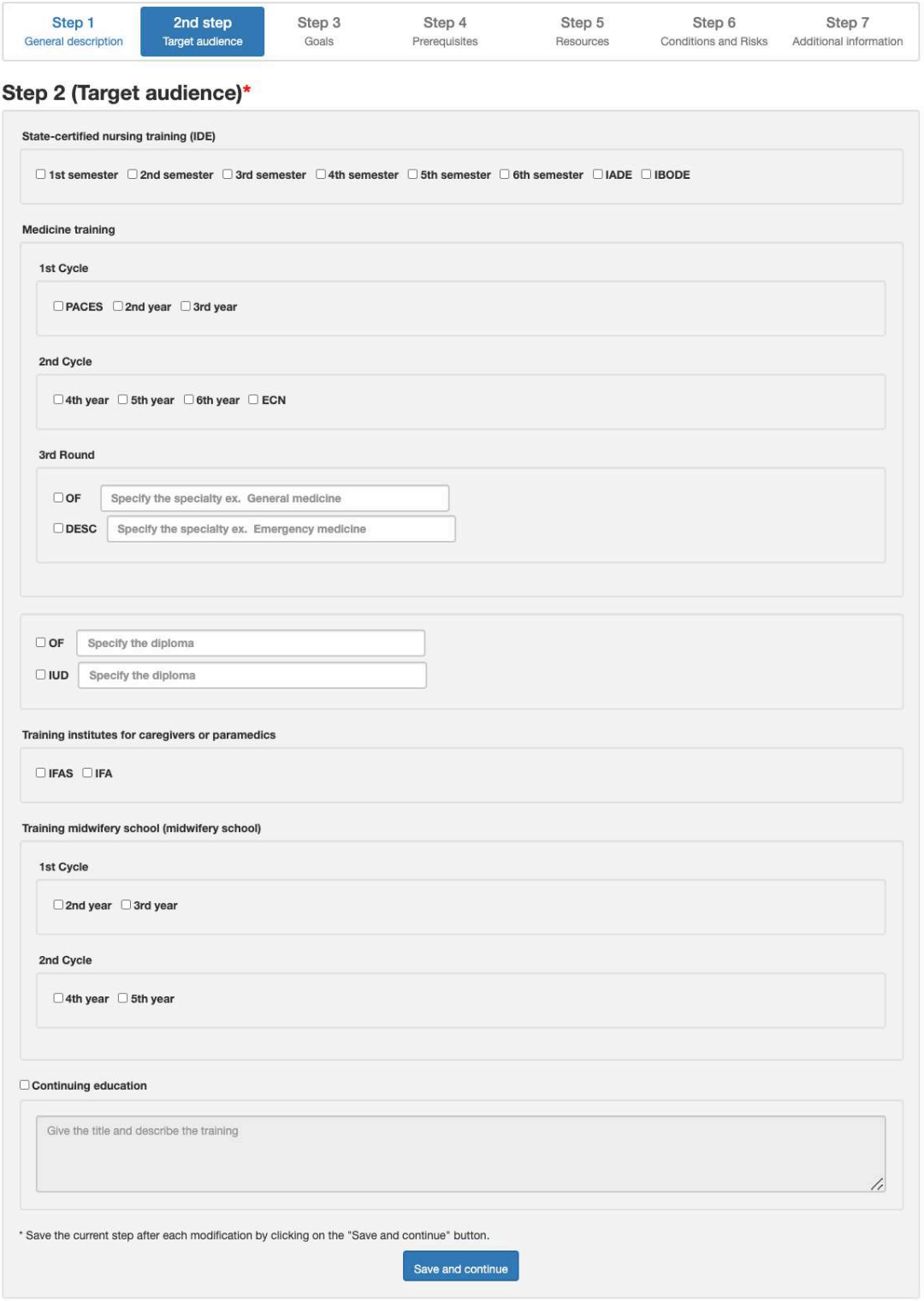

FIGURE A.2: Step 2 in the online questionnaire (target audience)

The third step of the questionnaire is shown in Figure A.3, where the health educator describes the objectives of the training session. The objectives of a session are defined in the context of the two following components: procedures, and assessments.

In the top part of the form, the expert fills in the "procedures", i.e., different actions in the training. A training session may have many procedures. Each procedure is described by a name, an objective (what needs to be learned by performing the action), description, and an optional resource (e.g., an explanatory image). In case the expert has second thoughts on a procedure and does not want to add it to the session after all, he/she will uncheck the "register" check box.

In the bottom part of the form, the expert provides information about the "assessment methods" of the training session. Each assessment method is identified with a type (e.g, multiple choice question, abbr., MCQ) and a link to the content of the assessment.

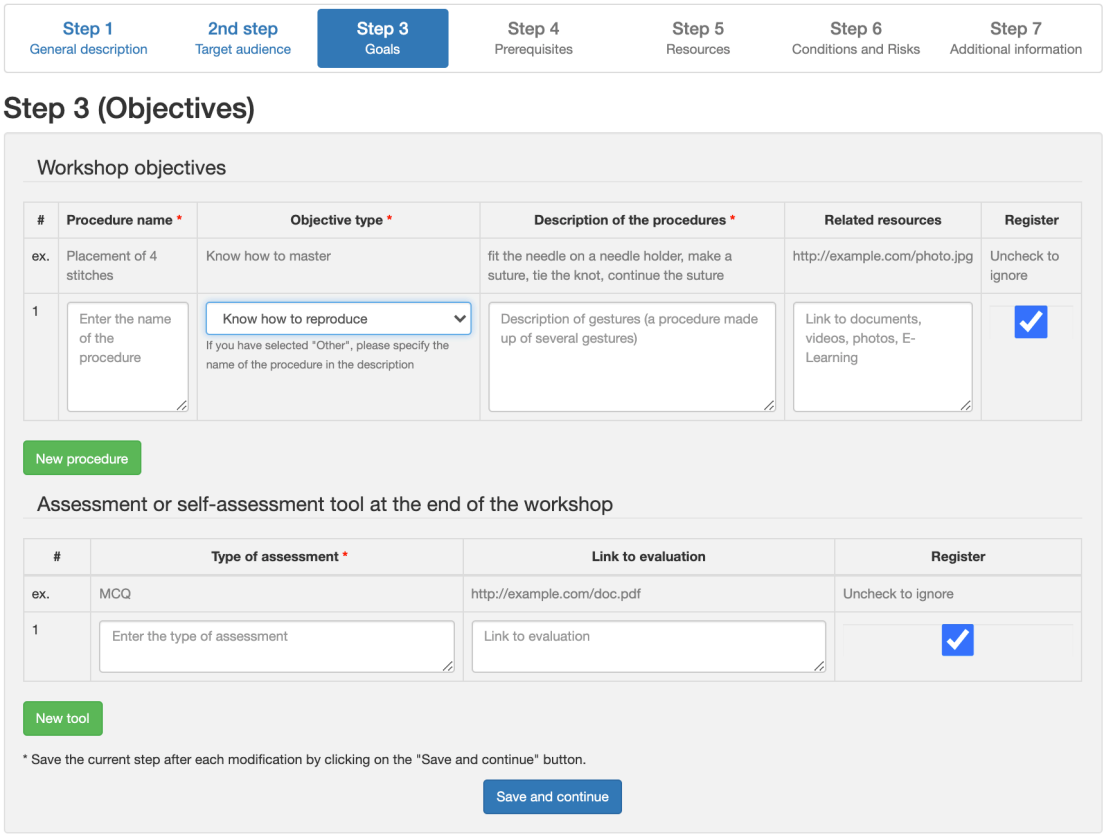

FIGURE A.3: Step 3 of the online questionnaire (goals)

Figure A.4 shows the fourth step of the online questionnaire, where the educator defines the prerequisites of the training session, i.e., what needs to be done before starting the training. Each prerequisite is described with a type, description, and an optional link for more explanation about the prerequisite. In case the expert has second thoughts on a prerequisite and does not want to add it to the session after all, he/she will uncheck the "register" check box.

The type of the prerequisite identifies its nature. For instance, it could be a video that has to be watched, another session to be validated, or a rule to know, before starting this session.

In the bottom part of the form, the expert describes the nature of assessments for the prerequisites. Similar to the previous step, each assessment is identified with a type and a link to its content.

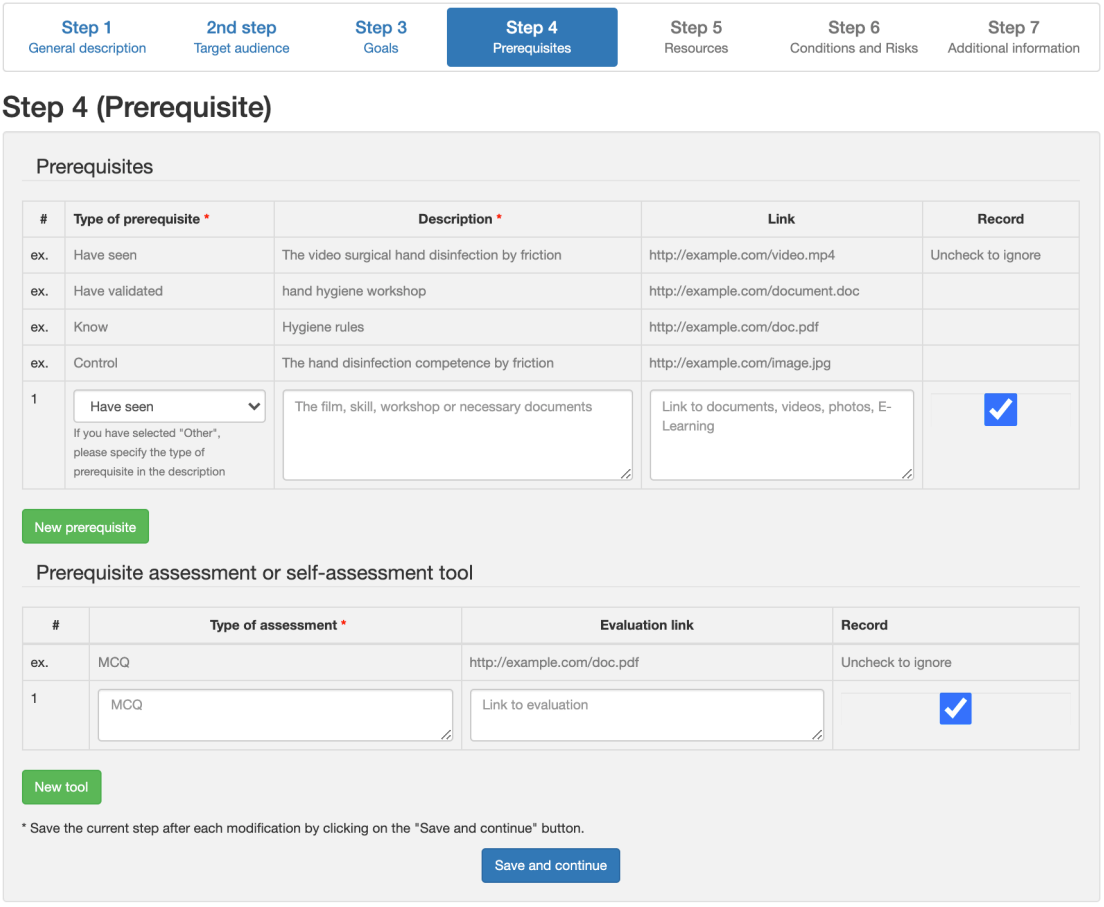

FIGURE A.4: Step 4 of the online questionnaire (prerequisites)

Figure A.5 shows the fifth step of the online questionnaire where the educator describes the resources required for the training session in four different categories: time, trainees, trainers, and material. First, the expert defines the time aspect of the resources as the minimum and maximum duration of the session in minutes. Second, he/she determines the minimum and maximum number of learners in the session. Third, the required human resources are defined, where each resource is described with a type (e.g., technician, assistant, etc.), a description, and the minimum and maximum number of the required resource. Last, materialistic resources are also described, such as consumables, simulation devices, etc.

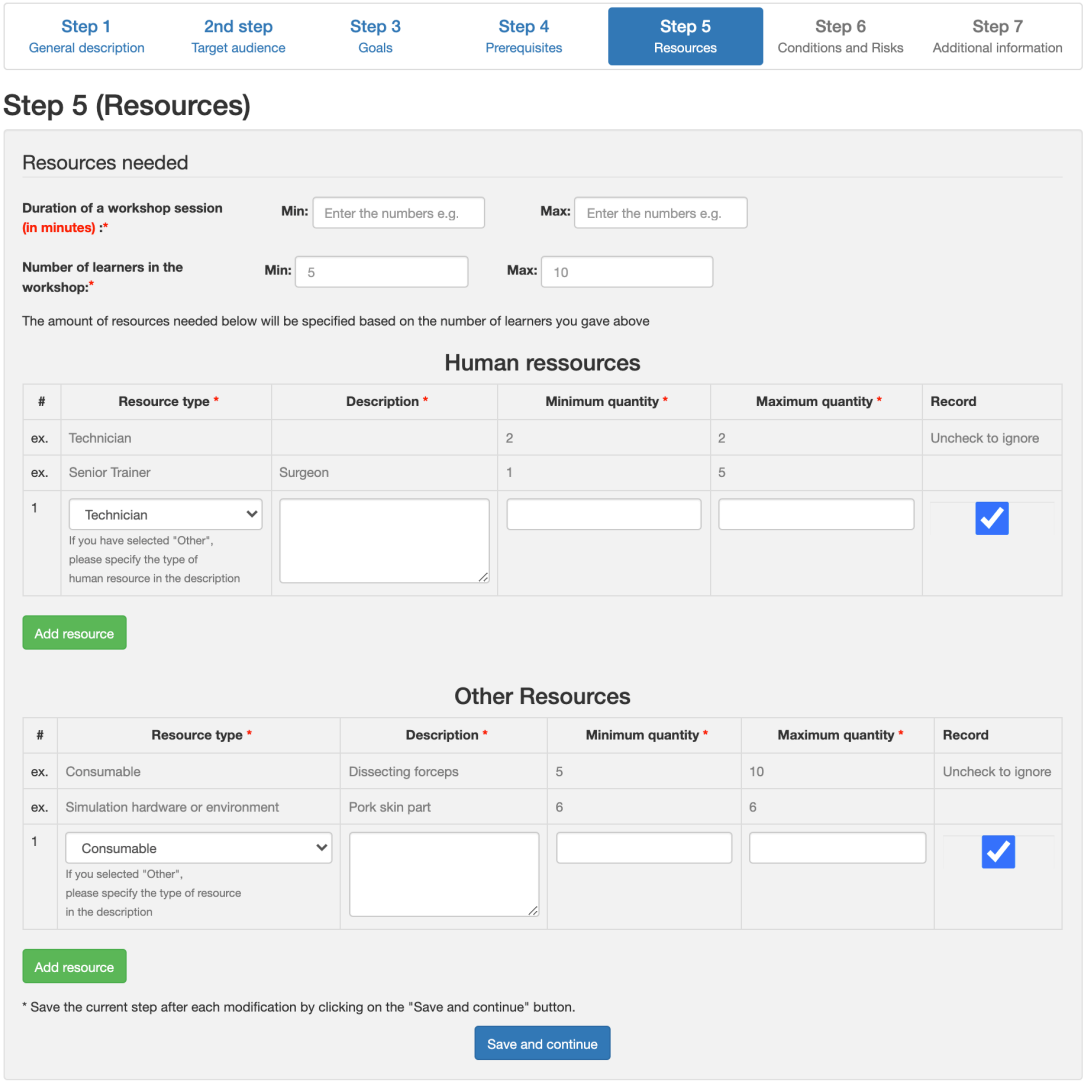

FIGURE A.5: Step 5 of the online questionnaire (resources)

Beyond preliminary information about a training session, it is also necessary to describe its associated terms and conditions, as well as its potential risks. Figure A.6 shows the sixth step of the questionnaire which captures this information.

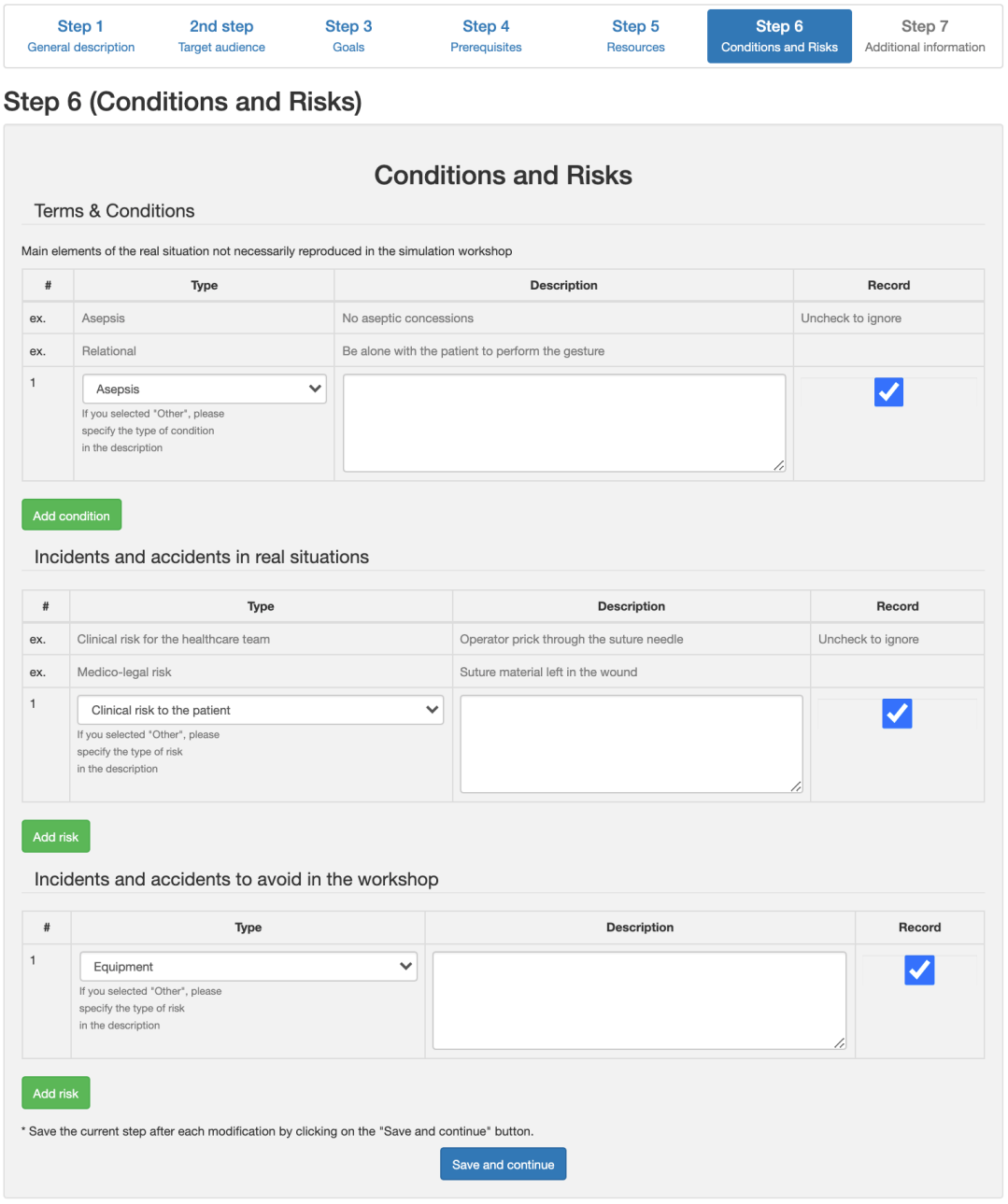

FIGURE A.6: Step 6 of the online questionnaire (conditions and risks)

In the last step of the questionnaire (Step 7 shown in Figure A.7), the expert can provide any other information about the training session that could not be fit in the previous steps. We also ask the educator if he/she is willing to be contacted for the next iterations of our study. The experts who participated in our expert study (Chapter 5) had all answered "yes" to this question.

Once the expert clicks the "finish" button in this form, a summary of all entered information will be shown, where he/she can verify the entries, and correct them if necessary.

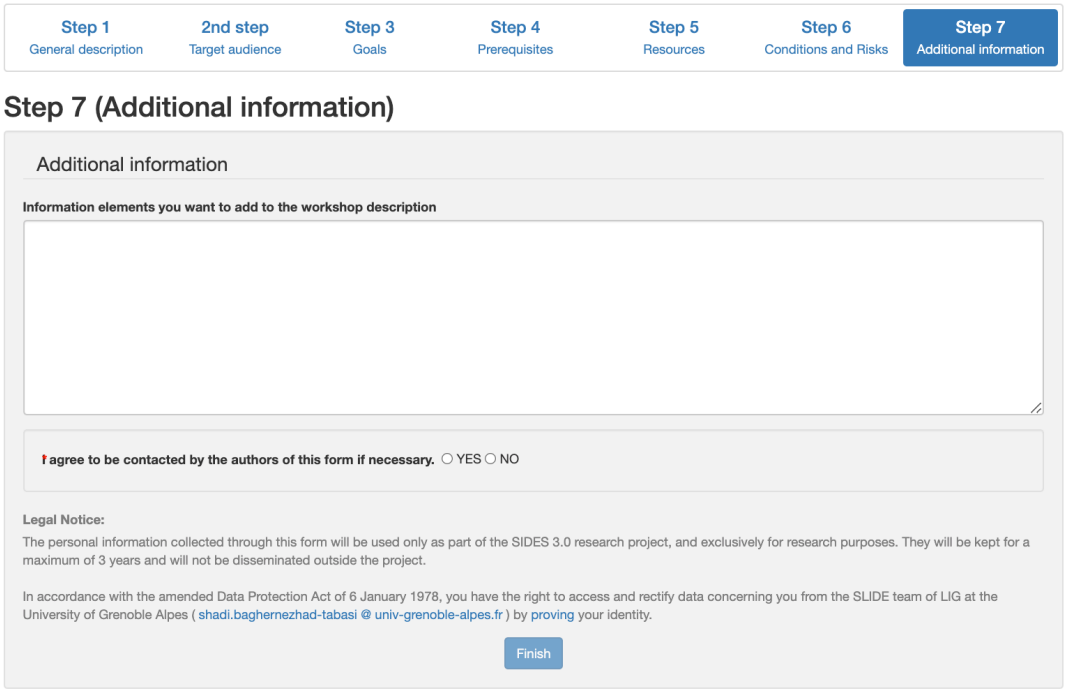

FIGURE A.7: Step 7 of the online questionnaire (additional information)

Appendix B

Mapping Rules

In Section 4.3.2.2, we introduced mapping rules as providers of the full RDF specification of the pre-filled Web pages that have to be created for a focus class. Among 16 mapping rules proposed in this thesis, 9 are already presented in Chapter 4. In this part of the appendix, we present the remaining mapping rules.

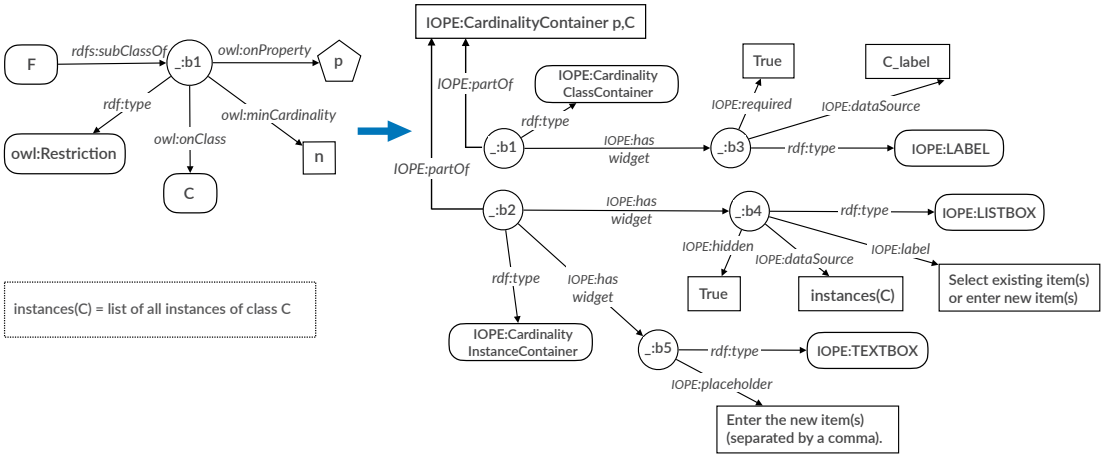

FIGURE B.1: Mapping rule #5, employed for a `cardinality` constraint where $subClasses(C)$ is empty, but $instances(C)$ is not empty, and $n > 0$.

■ **Mapping rule #5 for a cardinality restriction** (p `min` n C) **for** F **such that** $n > 0$, corresponds to the case where C does not have a hierarchy of subclasses, but has a list of instances in the domain ontology. This mapping rule is presented in Figure B.1. For the `IOPE:CardinalityClassContainer`, a widget of type `IOPE:LABEL` is created as a blank node with the property `IOPE:dataSource` filled by the label of class C, i.e., 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:Cardinality-InstanceContainer`, a widget of type `IOPE:LISTBOX` is created as a blank node with the property `IOPE:dataSource` filled with the list

instances(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)" to enable the experts to enter new instances.

■ **Mapping rule #6 for a cardinality restriction** (p `min` n C) **for** F **such that** $n > 0$, corresponds to the case where C has a hierarchy of sub-classes but does not have a list of instances in the domain ontology. This mapping rule is presented in Figure B.2. For the `IOPE:CardinalityClassContainer`, a widget of type `IOPE:TREEVIEW` is created as a blank node with the property `IOPE:dataSource` filled with 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 the experts to display the sub-class hierarchy, interactively. 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 enable the experts to enter new instances or provide a minimum required number.

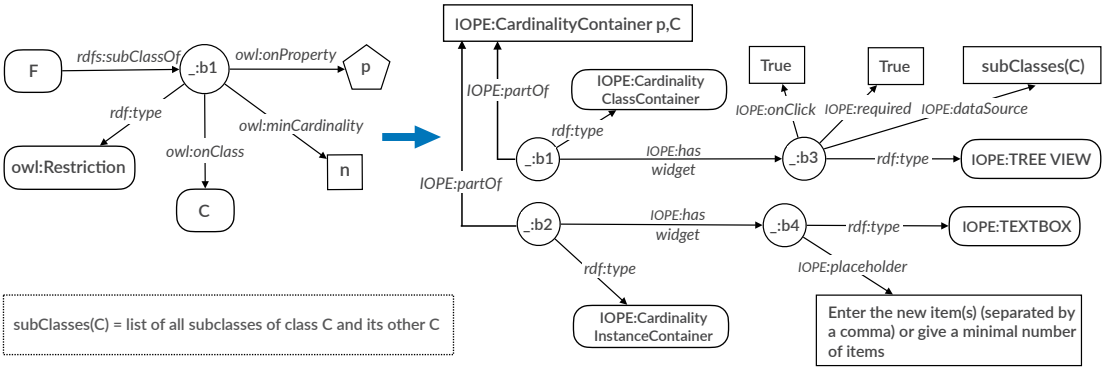

FIGURE B.2: Mapping rule #6, employed for a `cardinality` constraint where *subClasses*(C) is not empty, but *instances*(C) is empty, and $n > 0$.

■ **Mapping rule #9 for a cardinality restriction** (p `min` n C) **for** F **such that** $n = 0$, corresponds to the case where C does not have either a hierarchy of sub-classes or a list of instances in the domain ontology. This mapping rule is presented in Figure B.3. For `IOPE:CardinalityClassContainer`, a widget of type `IOPE:LABEL` is created as a blank node with the property `IOPE:dataSource` filled by the label of class C, i.e., "*C_label*". Given $n = 0$, the property `IOPE:required` is set to False for this widget, to indicate that it is up to the user to select a choice from this widget. As the class C does not have any instances for `IOPE:CardinalityInstanceContainer`, a widget

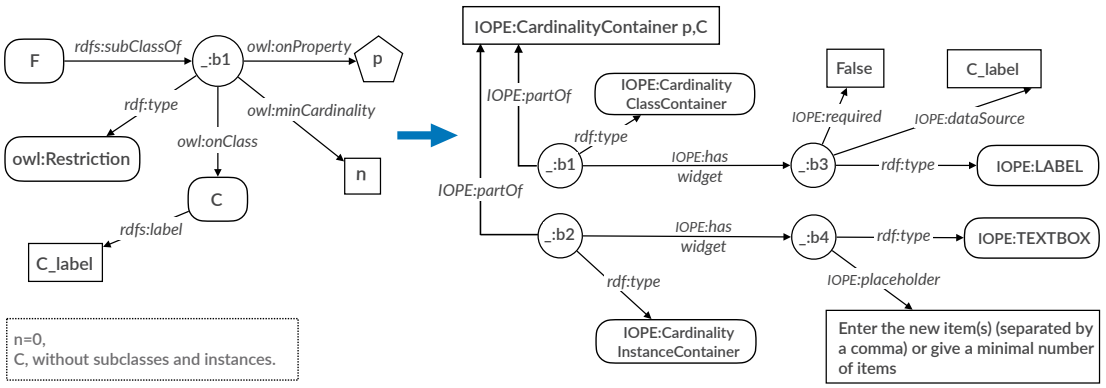

FIGURE B.3: Mapping rule #9, employed for a `cardinality` constraint where $subClasses(C)$ and $instances(C)$ are empty, and $n = 0$.

of the type `IOPE:TEXTBOX` is created with the `IOPE:placeholder` property set to the value "enter the new item(s) (separated by a comma) or provide a minimal number of items" in order to enable users to enter new instances, or provide a minimum number of requirements.

■ **Mapping rule #10 for a cardinality restriction** (p `min` n C) **for** F **such that** $n = 0$, corresponds to the case where C does not have a hierarchy of sub-classes, but has a list of instances in the domain ontology. This mapping rule is presented in Figure B.4. For the `IOPE:CardinalityClassContainer`, a widget of type `IOPE:LABEL` is created as a blank node with the property `IOPE:dataSource` filled by the label of class C, i.e., C_label. The property `IOPE:required` is set to False for this widget to indicate that it is up to the user to select a choice from this widget. For the `IOPE:CardinalityInstanceContainer`, a widget of type `IOPE:LISTBOX` is created as a blank node with the property `IOPE:dataSource` filled with the list $instances(C)$, 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)" to enable the experts to enter new instances.

■ **Mapping rule #11 for a cardinality restriction** (p `min` n C) **for** F **such that** $n = 0$, corresponds to the case where C has a hierarchy of sub-classes but does not have a list of instances in the domain ontology and is presented in Figure B.5. For the `IOPE:CardinalityClassContainer`, a widget of type `IOPE:TREEVIEW` is created as a blank node with the property `IOPE:dataSource` filled with 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` is set to False for this widget to indicate that it is up to the user to select a choice

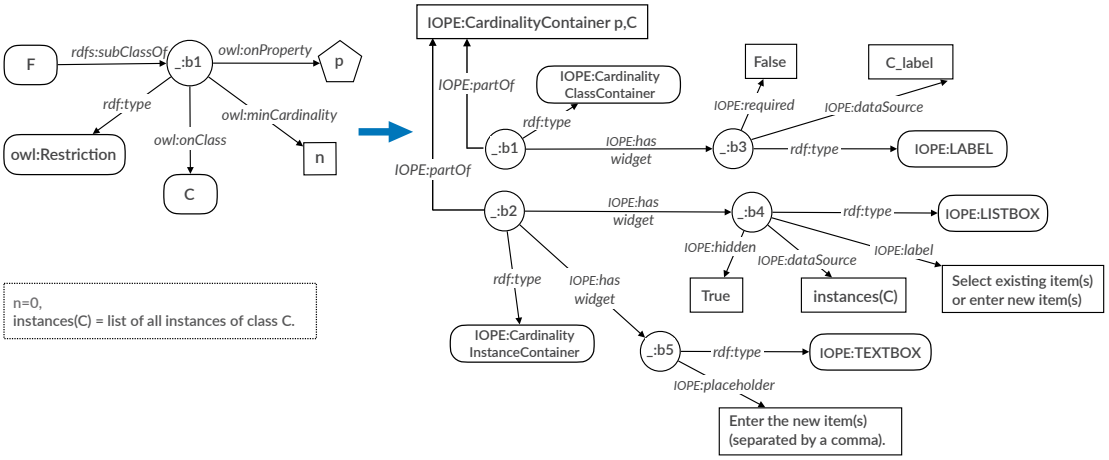

FIGURE B.4: Mapping rule #10, employed for a `cardinality` constraint where $subClasses(C)$ is empty, but $instances(C)$ is not empty, and $n = 0$.

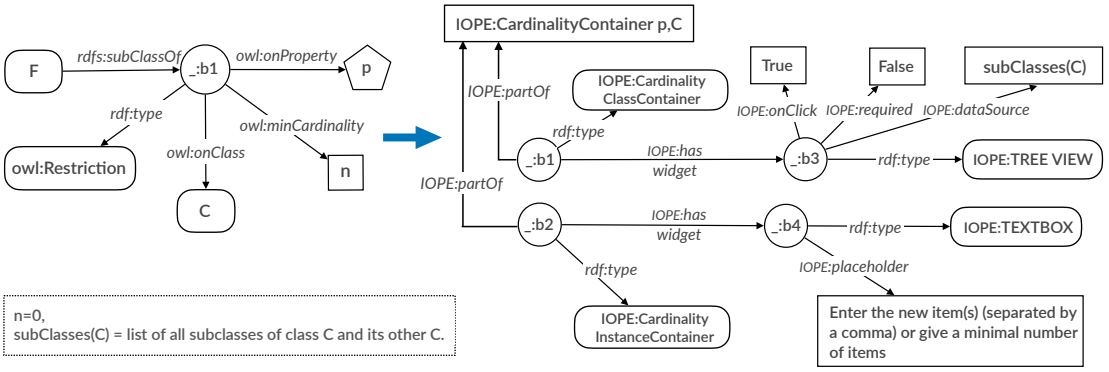

FIGURE B.5: Mapping rule #11, employed for a `cardinality` constraint where $subClasses(C)$ is not empty, but $instances(C)$ is empty, and $n = 0$.

from this widget. 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 enable the experts to enter new instances or provide a minimum required number.

■ **Mapping rule #14 for "domain and range" constraints for F on property p such that** (*p* `rdfs:domain` *F*) **and** (*p* `rdfs:range` *C*), corresponds to the case where the class range of the property p does not have a hierarchy of sub-classes, but has a list of instances in the domain ontology. Figure B.6 presents this mapping rule. In this case, the specific container "`IOPE:RangeContainer` p, C" is decomposed into two sub-containers defined as blank nodes, with types "`IOPE: RangeClassContainer`" and "`IOPE:RangeInstanceContainer`". For the `IOPE: RangeClassContainer`, a widget of type `IOPE:LABEL` is created as a blank node with the property `IOPE:dataSource` filled by the label of class C, i.e., 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:RangeInstanceContainer, a widget of type IOPE:LISTBOX is created as a blank node with the property IOPE:dataSource filled with the list *instances*(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)" to enable the experts to enter new instances.

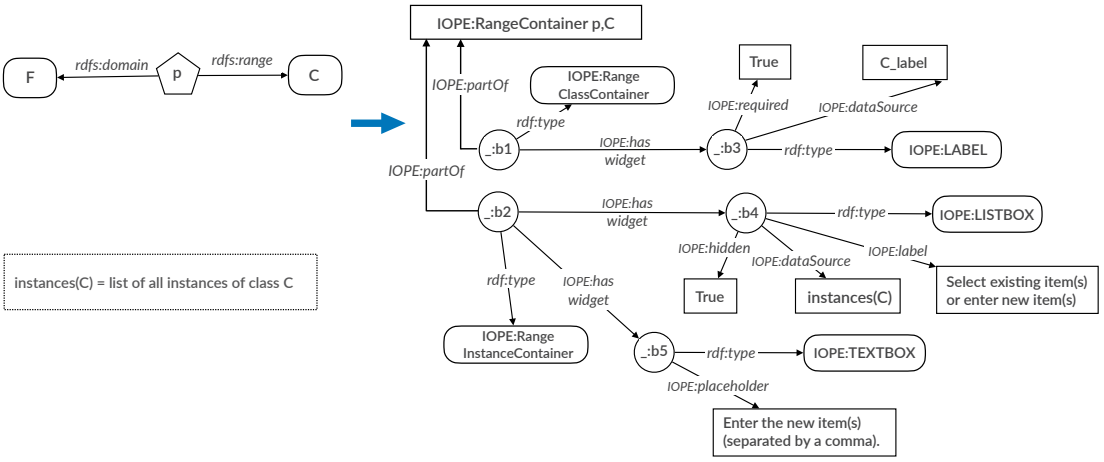

FIGURE B.6: Mapping rule #14, employed for "domain and range" constraints, where *subClasses*(C) is empty, but *instances*(C) is not empty.

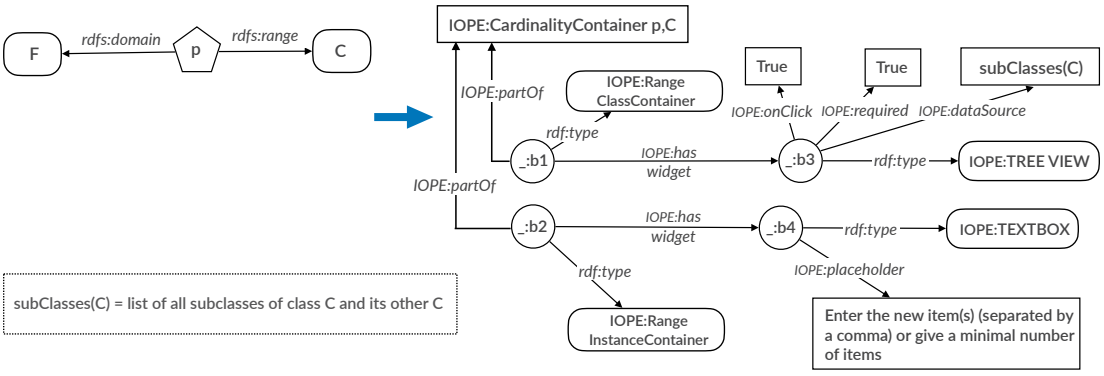

FIGURE B.7: Mapping rule #15, employed for "domain and range" constraints, where *subClasses*(C) is not empty, but *instances*(C) is empty.

■ **Mapping rule #15 for "domain and range" constraints for F on property p such that** (p IOPE:domain F) **and** (p IOPE:range C), corresponds to the case where the class range of the property p has a hierarchy of sub-classes but does not have a list of instances in the domain ontology. This mapping rule is presented in Figure B.7. For the IOPE:RangeClassContainer, a widget of type IOPE:TREEVIEW is created as a blank node with the property IOPE:dataSource filled with 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 the experts to display the sub-class hierarchy, interactively. For the IOPE:RangeInstanceContainer, 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 enable the experts to enter new instances or provide a minimum required number.

Bibliography

[AB18] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[ABGRA14] Ibrahim Ahmed Al-Baltah, Abdul Azim Abdul Ghani, Wan Nurhayati Wan Ab Rahman, and Rodziah Atan. A comparative study on ontology development methodologies towards building semantic conflicts detection ontology for heterogeneous web services. *Research Journal of Applied Sciences, Engineering and Technology*, 7(13):2674–2679, 2014.

[AH11] Dean Allemang and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[AMD+10] Rajesh Aggarwal, Oliver T Mytton, Milliard Derbrew, David Hananel, Mark Heydenburg, Barry Issenberg, Catherine MacAulay, Mary Elizabeth Mancini, Takeshi Morimoto, Nathaniel Soper, et al. Training and simulation for patient safety. *BMJ Quality & Safety*, 19(Suppl 2):i34–i43, 2010.

[ANR+19] Natanael Arndt, Patrick Naumann, Norman Radtke, Michael Martin, and Edgard Marx. Decentralized collaborative knowledge management using git. *Journal of Web Semantics*, 54:29–47, 2019.

[AORS15] Sihem Amer-Yahia, Behrooz Omidvar-Tehrani, Senjuti Basu Roy, and Nafiseh Shabib. Group recommendation with temporal affinities. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015*, pages 421–432. OpenProceedings.org, 2015.

[AT13] William Albert and Thomas Tullis. *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013.

[BBDW17] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. A taxonomy and survey of dynamic graph visualization. In *Computer Graphics Forum*, pages 133–159. Wiley Online Library, 2017.

[BHBL11] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

[BHLX13] AS Butt, A Haller, S Liu, and L Xie. Activeraul: Automatically generated web interfaces for creating rdf data. *Semantic Web*, 2013, 2013.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[BT21] Shadi Baghernezhad Tabasi. IOPE implementation. `https://github.com/shadi-tabasi/IOPE.git`, 2021.

[BTRD+19] Shadi Baghernezhad-Tabasi, Marie-Christine Rousset, Loïc Druette, Fabrice Jouanot, and Celine Meurger. OntoSAMSEI: Interactive ontology modeling for supporting simulation-based training in Medicine. In *IC3K*, 2019. KEOD Doctoral Consortium.

[CDH+17] Jean-Paul Calbimonte, Fabien Dubosson, Roger Hilfiker, Alexandre Cotting, and Michael Schumacher. The medred ontology for representing clinical data acquisition metadata. In *International Semantic Web Conference*, pages 38–47. Springer, 2017.

[CFLGP03] Oscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & knowledge engineering*, 46(1):41–64, 2003.

[DFP+20] Evanthia Dimara, Steven Franconeri, Catherine Plaisant, Anastasia Bezerianos, and Pierre Dragicevic. A task-based taxonomy of cognitive biases for information visualization. *IEEE Trans. Vis. Comput. Graph.*, 26:1413–1432, 2020.

[DLD08] Pieter De Leenheer and Christophe Debruyne. Dogma-mess: A tool for fact-oriented collaborative ontology evolution. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 797–806. Springer, 2008.

[DMDLM06] Aldo De Moor, Pieter De Leenheer, and Robert Meersman. Dogma-mess: A meaning evolution support system for interorganizational ontology engineering. In *International Conference on Conceptual Structures*, pages 189–202. Springer, 2006.

[DNM16] Antonio De Nicola and Michele Missikoff. A lightweight methodology for rapid ontology engineering. *Communications of the ACM*, 59(3):79–86, 2016.

[DPD16] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. Aide: an active learning-based approach for interactive data exploration. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2842–2856, 2016.

[ea20] Xin Luna Dong et al. Autoknow: Self-driving knowledge collection for products of thousands of types. In *KDD*, pages 2724–2734. ACM, 2020.

[EM06] Martha Evens and Joel Michael. *One-on-one tutoring by humans and computers*. Psychology Press, 2006.

[EW95] Chris Eliot and Beverly Park Woolf. An adaptive student centered curriculum for an intelligent training system. *User Modeling and User-Adapted Interaction*, 5(1):67–86, 1995.

[EW16] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48:1–4, 2016.

[FBC+17] Richard H Feins, Harold M Burkhart, John V Conte, Daniel N Coore, James I Fann, George L Hicks Jr, Jonathan C Nesbitt, Paul S Ramphal, Sharon E Schiro, K Robert Shen, et al. Simulation-based training in cardiac surgery. *The Annals of thoracic surgery*, 103(1):312–321, 2017.

[FCL+17] Yixiang Fang, Reynold Cheng, Siqiang Luo, Jiafeng Hu, and Kai Huang. C-explorer: Browsing communities in large graphs. *Proc. VLDB Endow.*, 10(12):1885–1888, 2017.

[FLGP02] Mariano Fernández-López and Asunción Gómez-Pérez. Overview and analysis of methodologies for building ontologies. *The knowledge engineering review*, 17(2):129, 2002.

[FLGPJ97] Mariano Fernández-López, Asunción Gómez-Pérez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. *American Association for Artificial Intelligence*, 1997.

[FO15] Laura Freina and Michela Ott. A literature review on immersive virtual reality in education: state of the art and perspectives. In *The international scientific conference elearning and software for education*, 2015.

[Gab04] David M Gaba. The future vision of simulation in health care. *BMJ Quality & Safety*, 13(suppl 1):i2–i10, 2004.

[GAVS11] Stephan Grimm, Andreas Abecker, Johanna Völker, and Rudi Studer. Ontologies and the semantic web. *Handbook of Semantic Web Technologies*, pages 507–579, 2011.

[GDMF12] Fausto Giunchiglia, Biswanath Dutta, Vincenzo Maltese, and Feroz Farazi. A facet-based methodology for the construction of a large-scale geospatial ontology. *Journal on data semantics*, 1(1):57–73, 2012.

[GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

[GPFLC06] Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer, 2006.

[GPSF09] Asunción Gómez-Pérez and Mari Carmen Suárez-Figueroa. Neon methodology for building ontology networks: a scenario-based methodology. *Demetra EOOD*, 2009.

[Gru93] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[Gru95] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.

[HBZC17] Kai Huang, Sourav S. Bhowmick, Shuigeng Zhou, and Byron Choi. PICASSO: exploratory search of connected subgraph substructures in graph databases. *Proc. VLDB Endow.*, 10(12):1861–1864, 2017.

[HFM07] Nathalie Henry, Jean-Daniel Fekete, and Michael J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1302–1309, 2007.

[HJ02] Clyde W Holsapple and Kshiti D Joshi. A collaborative approach to ontology design. *Communications of the ACM*, 45(2):42–47, 2002.

[HUH10] Armin Haller, Jürgen Umbrich, and Michael Hausenblas. Raul: Rdfa user interface language - A data processing model for web applications. In *WISE*, volume 6488, pages 400–410. Springer, 2010.

[IMM+13] Rizwan Iqbal, Masrah Azrifah Azmi Murad, Aida Mustapha, Nurfadhlina Mohd Sharef, et al. An analysis of ontology engineering methodologies: A literature review. *Research journal of applied sciences, engineering and technology*, 6(16):2993–3000, 2013.

[Ipe10] Panagiotis G Ipeirotis. Demographics of mechanical turk. *NYU working paper*, 2010.

[Kin68] Lester S. King. Signs and Symptoms. *JAMA*, 206, 1968.

[KP10] Konstantinos Kotis and Andreas Papasalouros. Learning useful kick-off ontologies from query logs: Hcome revised. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 345–351. IEEE, 2010.

[Krö17] Markus Krötzsch. Ontologies for knowledge graphs? In *Description Logics*, 2017.

[KV06] Konstantinos Kotis and George A Vouros. Human-centered ontology engineering: The hcome methodology. *Knowledge and Information Systems*, 10(1):109–131, 2006.

[KVS20] Konstantinos I Kotis, George A Vouros, and Dimitris Spiliotopoulos. Ontology engineering methodologies for the evolution of living and reused ontologies: status, trends, findings and recommendations. *The Knowledge Engineering Review*, 35, 2020.

[Lat10] Fatimah Lateef. Simulation-based learning: Just like the real thing. *Journal of Emergencies, Trauma and Shock*, 3(4):348, 2010.

[LNHE16] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Visualizing ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016.

[Lor18] Steven Loria. textblob documentation. *Release 0.15*, 2, 2018.

[LT+75] Harold A Linstone, Murray Turoff, et al. *The delphi method.* Addison-Wesley Reading, MA, 1975.

[MFC+17] Pierre Maillot, Sébastien Ferré, Peggy Cellier, Mireille Ducassé, and Franck Partouche. Nested forms with dynamic suggestions for quality RDF authoring. In *DEXA*, volume 10438, pages 35–45. Springer, 2017.

[MPE+15] D. Mouromtsev, D. Pavlov, Yury Emelyanov, A. Morozov, Daniil Razdyakonov, and M. Galkin. The simple web-based tool for visualization and sharing of semantic data and ontologies. In *International Semantic Web Conference*, 2015.

[NFG+20] Eirini Ntoutsi, Pavlos Fafalios, Ujwal Gadiraju, Vasileios Iosifidis, Wolfgang Nejdl, Maria-Esther Vidal, Salvatore Ruggieri, Franco Turini, Symeon Papadopoulos, Emmanouil Krasanakis, et al. Bias in data-driven

artificial intelligence systems—an introductory survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(3):e1356, 2020.

[NGJ+19] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges. *Communications of the ACM*, 62 (8):36–43, 2019.

[NJ17] Ingrid Nunes and Dietmar Jannach. A systematic review and taxonomy of explanations in decision support and recommender systems. *User Modeling and User-Adapted Interaction*, 27(3):393–444, 2017.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9, 2001.

[NSD+01] Natalya Fridman Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and Mark A. Musen. Creating semantic web contents with protégé-2000. *IEEE Intell. Syst.*, 16(2):60–71, 2001.

[OA19] Behrooz Omidvar-Tehrani and Sihem Amer-Yahia. Data pipelines for user group analytics. In *SIGMOD Conference*, pages 2048–2053. ACM, 2019.

[PCHK20] Irene Polikoff, Robert Coyne, Ralph Hodgson, and Holger Knublauch. Topquadrant topbraid composer. `https://www.topquadrant.com/products/topbraid-composer/`, 2020. Accessed: 2021-01-15.

[PJN+19] Olivier Palombi, Fabrice Jouanot, Nafissetou Nziengam, Behrooz Omidvar-Tehrani, Marie-Christine Rousset, and Adam Sanchez. Ontosides: Ontology-based student progress monitoring on the national evaluation system of french medical schools. *Artif. Intell. Medicine*, 96:59–67, 2019.

[PMMB88] Joseph Psotka, Leonard Daniel Massey, Sharon A Mutter, and John Seely Brown. *Intelligent tutoring systems: Lessons learned*. Psychology Press, 1988.

[PWC+17] Zhifei Pang, Sai Wu, Gang Chen, Ke Chen, and Lidan Shou. Flashview: An interactive visual explorer for raw data. *Proc. VLDB Endow.*, 10(12):1869–1872, 2017.

[RJN20] Protiva Rahman, Lilong Jiang, and Arnab Nandi. Evaluating interactive data systems. *VLDB J.*, 29(1):119–146, 2020.

[RSH+16] Thomas Rebele, Fabian Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *International semantic web conference*, pages 177–185. Springer, 2016.

[SA11] R Subhashini and J Akilandeswari. A survey on ontology construction methodologies. *International Journal of Enterprise Computing and Business Systems*, 1(1):60–72, 2011.

[Set09] Burr Settles. Active learning literature survey. *University of Wisconsin-Madison Department of Computer Sciences*, 2009.

[SF10] Mari Carmen Suárez-Figueroa. *NeOn Methodology for building ontology networks: specification, scheduling and reuse.* PhD thesis, Informatica, 2010.

[SJH+07] Christophe Segouin, Jean Jouquan, Brian Hodges, Pierre-Henri Bréchat, Stéphane David, Dominique Maillard, Benoit Schlemmer, and Dominique Bertrand. Country report: medical education in france. *Medical Education*, 41(3):295–301, 2007.

[SKS+16] Anjali Sehrawat, Robert Keelan, Kenji Shimada, Dona M Wilfong, James T McCormick, and Yoed Rabin. Simulation-based cryosurgery intelligent tutoring system prototype. *Technology in cancer research & treatment*, 15(2):396–407, 2016.

[SLR14] Elena Simperl and Markus Luczak-Rösch. Collaborative ontology engineering: a survey. *The Knowledge Engineering Review*, 2014.

[SSD13] Bernd Stadlhofer, Peter Salhofer, and Augustin Durlacher. An overview of ontology engineering methodologies in the context of public administration. In *Proceedings of the 7th International Conference on Advances in Semantic Processing, IARIA, Porto, Portugal*, volume 29, pages 36–42, 2013.

[SSSS01] Steffen Staab, Rudi Studer, H-P Schnurr, and York Sure. Knowledge processes and ontologies. *IEEE Intelligent systems*, 16(1):26–34, 2001.

[ST06] Elena Paslaru Bontas Simperl and Christoph Tempich. Ontology engineering: A reality check. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 836–854. Springer, 2006.

[STM+18] Angelo A Salatino, Thiviyan Thanapalasingam, Andrea Mannocci, Francesco Osborne, and Enrico Motta. The computer science ontology:

a large-scale taxonomy of research areas. In *International Semantic Web Conference*, pages 187–205. Springer, 2018.

[TDJ+21a] Shadi Baghernezhad Tabasi, Loic Druette, Fabrice Jouanot, Celine Meurger, and Marie-Christine Rousset. Iope: Interactive ontology population and enrichment. *SEMANTiCS*, 2021.

[TDJ+21b] Shadi Baghernezhad Tabasi, Loic Druette, Fabrice Jouanot, Celine Meurger, and Marie-Christine Rousset. Ontosamsei: Interactive ontology engineering for supporting simulation-based training in medicine. In *30th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2021, Virtual Event, France, October 27-29, 2021*. IEEE, 2021.

[Tho05] James J Thomas. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society, 2005.

[UG+96] Michael Uschold, Michael Gruninger, et al. Ontologies: Principles, methods and applications. *University of Edinburgh Artificial Institute Applications Institute*, 1996.

[UK95] Michael Uschold and Martin King. *Towards a methodology for building ontologies*. Citeseer, 1995.

[VPCAR20] Andre Valdestilhas, Gustavo Publio, Andrea Cimmino Arriaga, and Thomas Riechert. Voceditor: An integrated environment to visually edit, validate and versioning rdf vocabularies. In *International Conference on Semantic Computing (ICSC)*, 12 2020.

[VPTS05] Denny Vrandečić, Sofia Pinto, Christoph Tempich, and York Sure. The diligent knowledge processes. *Journal of Knowledge Management*, 2005.

[WHA07] Wesley Willett, Jeffrey Heer, and Maneesh Agrawala. Scented widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, 2007.

[WLA18] Vitalis Wiens, Steffen Lohmann, and Sören Auer. Webvowl editor: Device-independent visual ontology modeling. In *ISWC 2018 Posters & Demonstrations*, volume 2180 of *CEUR Workshop Proceedings*, 2018.

[WMH+20] Jesse Wright, Sergio José Rodríguez Méndez, Armin Haller, Kerry Taylor, and Pouya Ghiasnezhad Omran. Schímatos: A shacl-based web-form generator for knowledge graph editing. In *ISWC*, volume 12507, pages 65–80. Springer, 2020.

[YNDJ16] Usha Yadav, Gagandeep Singh Narula, Neelam Duhan, and Vishal Jain. Ontology engineering and development aspects: a survey. *International Journal of Education and Management Engineering*, 6(3):9–19, 2016.

[Zac07] Manuel Zacklad. Classification, thésaurus, ontologies, folksonomies: comparaisons du point de vue de la recherche ouverte d'information (roi). In *CAIS/ACSI 2007, 35e Congrès annuel de l'Association Canadienne des Sciences de l'Information. Partage de l'information dans un monde fragmenté: Franchir les frontières, sous la dir. de C. Arsenault et K. Dalkir. Montréal: CAIS/ACSI, 2007*, 2007.

[ZCA+18] Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W Bruce Croft. Towards conversational search and recommendation: System ask, user respond. In *Proceedings of the 27th acm international conference on information and knowledge management*, pages 177–186, 2018.

[ZMA19] Yongfeng Zhang, Jiaxin Mao, and Qingyao Ai. Sigir 2019 tutorial on explainable recommendation and search. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1417–1418, 2019.