

Ontology-Mediated Framework for Generating Answerable Questions and Feedback from Ontologies

Toky Hajatiana Raboanary^{1,2}[0000-0001-6133-4643] and C. Maria Keet^{1,3,4}[0000-0002-8281-0853]

¹ University of Cape Town, South Africa, traboanary@cs.uct.ac.za

² Centre for Artificial Intelligence Research (CAIR), South Africa

³ Stellenbosch University, South Africa

⁴ Meaningfy SARL, Lintgen, Luxembourg, maria.keet@meaningfy.ws

Abstract. Question and feedback generation from ontologies has appeal for the high-quality content it can provide, and to alleviate the test setting and marking burden of teachers and quiz question setters. Existing approaches lack a formal and expressive technique, however, which limits their ability to represent and process complex and semantically rich questions, answers, and explanatory feedback patterns. They typically require post-processing to eliminate irrelevant outputs, rely on entity labels rather than represented semantics, and they often lack modularity, reducing scalability and reusability. We solve this by focusing on the content determination stage in the natural language generation pipeline, and any realiser of choice to be used afterwards. We propose a framework consisting of an OWL 2 DL-based model combined with an optimised determination algorithm to efficiently fetch relevant content from the ontology. The framework was implemented and evaluated on content selection performance with several ontologies.

Keywords: Ontology-Driven Content Generation · Ontology-Based Question Generation · Natural Language Processing for semantic sources

1 Introduction

The automatic generation of questions and feedback from ontologies receives considerable interest from the perspectives of question and answer generation for educational questions [15,18], for retrofitting competency questions on ontologies [2], validating ontologies [1], and possibly also ontology-based explainability.

A typical ingredient of the approach is axiom patterns (types of axioms) to represent the knowledge for the questions and answers in such ontology-based question generation systems [1,8,18,22,24,25,26,30,31]. Their specific representation for modelling question types and feedback, however, is either not listed explicitly [1,18,22,30,31] or is tailored to a particular inexpressive logic, such as OWL Lite [1] or OWL 2 EL [18]. Even then, they may still use only the entity names or labels, without considering the full logical semantics of the entities

[1,8,18,24,30,31], or use only one simple class expression per question and answer [1,8,18,22,25,30,31]. Alternatively, SPARQL or SWRL rules are used [5,16,17]. Consequently, the models proposed in one paper are not usable or interoperable with that of others, causing existing results to remain largely isolated.

In addition, depending on the generation approach, it may require filtering out of irrelevant or unnecessary questions and feedback [1,5,8,18,24,25,30,31], or they may embed pattern logic directly into algorithms and thereby reducing scalability and reusability [8,25,26], affecting the ability to design complex questions, their correctness, and the scalability and reusability of doing so. Yet, unlike other natural language generation systems that face the typical problem of multiple input formats in the content selection stage [27], in this case, we always do have an ontology serialised in OWL.

To solve these issues, we devised a novel content selection framework for generating questions and feedback from ontologies that addresses the aforementioned limitations. The framework comprises: (a) a metamodel represented in OWL 2 DL to represent types of questions and feedback, and (b) an optimised content-determination algorithm that identifies all concrete axioms matching the patterns of the types of questions and feedback. Figure 1 provides an overview of the framework and presents the workflow, and delineates the contribution boundaries. From the metamodel, a number of models can be instantiated, where each model represents a type of question and its associated feedback. These are specified through axiom patterns containing variables (classes and object properties) that will be instantiated with concrete ontology entities. In the context of Description Logics (DL), an axiom pattern is a generalised template that represents recurrent forms of axioms; e.g., $\text{Child} \sqsubseteq \text{Person}$ instantiates the axiom pattern $C \sqsubseteq D$ (see Table 1). Then, a set of axiom patterns forms an axiom pattern set (APS), a question and feedback prerequisite, also called simply a prerequisite, that is relevant to a type of question and its feedback. Models are fed into the content selection algorithm, which outputs the concrete axioms extracted from the input ontology, from which we generate the questions and feedback. This output can be passed on to one’s preferred verbaliser, such as SimpleNLG [10], an LLM, or other realiser of choice (see [9,20] for recent reviews), to generate natural language questions, answers, and feedback. A brief illustration is included in the following example.

Example 1. As shown in Table 1, a model is defined by its axiom pattern set (APS), a prerequisite, and specific variable constraints. Concrete axioms can then be derived from the model. In this case, the input is the Family ontology. Table 1 lists two simple controlled language templates, although they also could be grammar-based. The algorithm searches the ontology to extract the relevant set, or sets, of axioms. A verbaliser processes the output. \diamond

The content extraction algorithm is shown to be of $\mathcal{O}(n^p)$ worst-case complexity, and a lower bound of $\Omega(n)$, where p is the number of axiom patterns constituting a prerequisite (in Example 1, $p = 4$). We implemented the metamodel and the algorithm in a proof-of-concept implementation to validate the theory, and stress-tested it with a number of models exploring the solution space

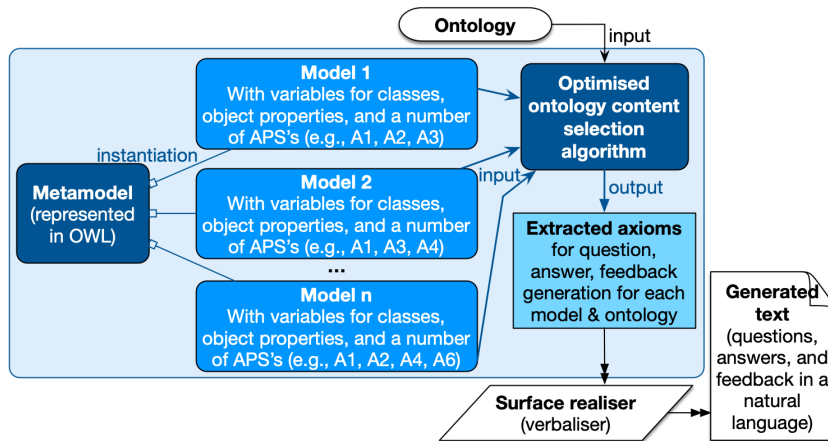


Fig. 1. Overview of the proposed ontology-based content selection framework, highlighted in blue. Models are instantiated from the metamodel, and the algorithm takes both the ontology and the models as input to produce all axioms usable for generating the selected types of questions, answers, and feedback. This output is then fed into one’s preferred verbaliser to generate the text.

and its effects with two ontologies: the African Wildlife Ontology (AWO) [13] and Exercise Medicine Ontology (EXMO) [19]. The results show that the optimised (cf. brute force) algorithm has an average relative speed-up of 518600,13.

This paper is organised as follows. Section 2 reviews the related work. This is followed by the metamodel in Section 3. Section 4 describes the optimised algorithm and provides both theoretical and empirical analyses. We discuss in Section 5 and conclude in Section 6. The supplementary materials are available at <https://github.com/toky-raboanary/FrameworkQFgen>.

2 Related work

A common approach to represent the content for questions, answers, and feedback abstractly, is to use axiom patterns in some way [1,8,18,22,26,30,31], and mostly only for lightweight ontologies represented in RDFS or OWL 2 EL [1], \mathcal{AL} [30], or \mathcal{EL} [18]. Alternatively, SPARQL is used to model the questions [5,16,17], where it is primarily used for retrieving RDF data and is not well-suited to highly expressive logic features, working with less than OWL 2 RL [5] or just RDFS [16,17], possibly with a ‘Knowledge Graph-like’ model to represent both question prerequisites and linguistic templates that combine machine learning and semantic-based techniques [17]. An approach with SWRL rules for MCQ generation [14] and object-oriented knowledge bases with answer set programming [6,7] also exist. Approaches to generate questions may produce inappropriate or invalid outputs [1,5,8,18,22,24,25,30,31]. This can be mitigated by applying a post-processing step, such as a *stem-exclusion* module [18], to filter out unsuit-

Table 1. Example of question, answer, and feedback generation, which is applied to a ‘traditional Family ontology’ with some extensions, such as property chains, and constraints, such as that class variables C, D, E and F are not a subclass of `dolce:Perdurant`.

Example 1	input ontology $O_1 = \text{Family}$, number of patterns $p = 4$, DL: \mathcal{SRQ}
Constraints	$C \sqsubseteq \neg\text{Perdurant}$, $D \sqsubseteq \neg\text{Perdurant}$, $E \sqsubseteq \neg\text{Perdurant}$, $F \sqsubseteq \neg\text{Perdurant}$
Axiom pattern set (APS)	Matching concrete axioms (CA) in O_1
$R1 \circ R2 \sqsubseteq R3$	$\text{hasFather} \circ \text{hasBrother} \sqsubseteq \text{hasUncle}$
$C \sqsubseteq D$	$\text{Child} \sqsubseteq \text{Person}$
$C \sqsubseteq \exists R2.F$	$\text{Child} \sqsubseteq \exists \text{hasParent.Father}$
$F \sqsubseteq \exists R3.E$	$\text{Father} \sqsubseteq \exists \text{hasBrother.Uncle}$
Templates	Sample generated text
Question template: Which [D][R3][E]?	Which person has an uncle?
Feedback template: The answer is [C], because [C] is [D], [C][R1][F], and [F][R2][E].	The answer is a child because a child is a person, a child has a father, and a father has a brother who is his uncle.

able questions, but such post-processing adds complexity and hinders scalability. One can incorporate filtering directly, as in the *question-card* model [24], where question types and feedback are represented, but their approach does not yet leverage the logic; consequently, numerous cases still require handling outside the model itself.

When axiom patterns are used to represent prerequisites, their placeholders are instantiated using only the ontology’s vocabulary rather than the full semantics of those elements [1,8,18,22,26,25,24,30,31]. Yet, not using the full semantics of the elements may result in confusion between the prerequisites and the input ontology. In several papers, however, the underlying modelling strategies are not clearly described [1,18,22,30,31], making it difficult to determine whether the same limitation applies. In axiom pattern-based approaches relying on OWL [1,8,18,22,25,26,24,30,31], the number of patterns within an axiom pattern set is a factor that determines the complexity of a model, or: what sort of questions can be generated and how elaborate the feedback can be (see Table 3 and Table 4 for illustration). This number is at most three in [22,25,26,30], whereas the number of patterns for one combination of question, answer, and feedback is unlimited in the model of [24], and unclear in other proposals [1,8,18,31].

Some studies embed the business logic and axiom patterns directly into algorithms or hard-coded generation procedures [8,25,26], which reduces the flexibility, scalability, and reusability of their approach. The other comparable studies [1,18,22,30,31] did not explicitly explain how the logic is integrated, and it is therefore unclear whether they ensure scalability and reusability. As far as we know, only [24] supports scalability and reusability through its architecture.

Summarising the limitations of existing approaches, they are:

- L1:** A lack of formal representation for question types feedback that supports the full features of OWL 2 DL [1,31,30,18,22,25,25,8,24,5];

Table 2. Description of the metamodel $\mathcal{M} = M_{\text{var}} \cup M_{\text{preq}}$. M_{var} and M_{preq} are to specify the variables and prerequisites, respectively. ‘representative’ means that the entity represents a variable in M_{var} .

M_{var}	M_{preq}
VarClass \sqsubseteq owl:Thing	VarClass \sqsubseteq owl:Thing {representative}
VarOP \sqsubseteq owl:topObjectProperty	VarOP \sqsubseteq owl:topObjectProperty {representative}
	Prerequisite \sqsubseteq owl:Thing ,
	Prerequisite \sqcap VarClass $\equiv \perp$

- L2:** The generation of irrelevant or unnecessary questions and feedback that requires additional filtering [1,31,30,18,25,8,24,5];
- L3:** Processing of variables solely based on element names or labels, without considering the full semantics of the entities (i.e., not including the axioms they participate in) [1,31,30,18,25,25,8,24];
- L4:** No evidence of the ability to model question and feedback types involving multiple axiom types, which are essential for representing complex, structurally rich patterns [1,31,30,18,22,25,25,8];
- L5:** Embedding pattern logic directly into algorithms, reducing scalability and reusability [8,25,26].

3 A metamodel to represent question types and feedback

Question and feedback types can be modelled using axiom patterns, SPARQL, or specialised representations. However, SPARQL provides limited support for expressing and reasoning over the complex logical constructs found in expressive DLs [3]. Using specialised representations, such as those in [6,14,17], constrains the modelling process to the structure and assumptions of those approaches; e.g., Kusuma et al. [17] restrict it to RDFS. Therefore, we use axiom patterns to fully support reasoning capabilities and exploit the expressivity of OWL 2 DL.

3.1 The metamodel for question and feedback specification

Separating the variables used to describe prerequisites from the axiom patterns has been shown to be beneficial [24]. Therefore, we follow the same spirit, with the difference that we leverage OWL 2 DL to define both the variables and the prerequisites. Our metamodel \mathcal{M} consists of two OWL ontologies, or modules: one for defining the variables (M_{var}) and the other for the prerequisites (M_{preq}), with $\mathcal{M} = M_{\text{var}} \cup M_{\text{preq}}$, as follows (the details of the components are discussed afterwards). Table 2 shows the main structure of the metamodel \mathcal{M} .

M_{var} : Definitions of the variables. M_{var} is part of the metamodel \mathcal{M} represented in an OWL ontology, where each variable may represent either a class or an object property, M_{var} is defined as follows:

- A designated class VarClass (a direct subclass of owl:Thing) is used to define variables of the type class.

- A designated object property **VarOP** (a direct subproperty of `owl:TopObjectProperty`), is used to define variables of type object property.

For classes, each variable class C of type class is declared as a subclass of **VarClass**, i.e., $C \sqsubseteq \text{VarClass}$. Each variable class C can be annotated with one of the following tags: *OnlyMe*, *MeAndChildren* or *OnlyChildren*. Let $\text{Cls}(O_1)$ denote the set of classes in the input ontology O_1 . If C is tagged as *OnlyMe*, the set of all valid classes constrained by C is defined as $S_C = \{C\}$. If C is tagged as *MeAndChildren*, the set of all valid classes constrained by C is defined as $S_C = \{c \in \text{Cls}(O_1) \mid c \sqsubseteq C\}$. If C is tagged as *OnlyChildren*, the set of all valid classes constrained by C is defined as $S_C = \{c \in \text{Cls}(O_1) \mid c \sqsubseteq C\} \setminus C$.

For object properties, any variable R representing an object property is declared as a subproperty of **VarOP**, i.e., $R \sqsubseteq \text{VarOP}$. The set of all valid object properties constrained by R is defined as $S_R = \{r \in \text{OP}(O_1) \mid r \sqsubseteq R\}$, where $\text{OP}(O_1)$ denotes the set of object properties in the input ontology O_1 .

Each entity in M_{var} can be mapped to, or matched with, an entity in different ontologies, such as an entity belonging to O_1 or an entity in a foundational ontology (e.g., DOLCE [23]), thereby providing semantics for the variables rather than treating them as bare placeholders. Let F be an ontology such that $F \notin \mathcal{M}$. The semantics of C and R , provided through their equivalence with entities in F , are defined as follows:

$$\exists x \in \text{Cls}(F) \text{ such that } C \equiv x, \text{ and } \exists y \in \text{OP}(F) \text{ such that } R \equiv y.$$

In Table 1, we illustrated an example of restriction by assuming that the variables C , D , E , and F are not *Perdurant*, meaning that they represent classes that do not unfold over time. In this context, *Perdurant* in the metamodel \mathcal{M} is equivalent to *Perdurant*'s DOLCE category ($\text{Perdurant}_{\mathcal{M}} \equiv \text{Perdurant}_{\text{DOLCE}}$).

M_{preq} : Definitions of the prerequisites. M_{preq} is part of the metamodel \mathcal{M} represented in an OWL ontology and is defined as follows:

- A designated class **VarClass**, declared as a direct subclass of `owl:Thing`, is used to represent variable classes in M_{preq} . Each class C' , where $C' \sqsubseteq \text{VarClass}$, serves as a representative variable for a variable $C \in M_{\text{var}}$, such that

$$\exists C \in M_{\text{var}} \text{ with } \text{name}(C') = \text{name}(C).$$

- A designated object property **VarOP**, declared as a direct subproperty of `owl:TopObjectProperty`, is used to represent variable object properties in M_{preq} . Each object property R' , where $R' \sqsubseteq \text{VarOP}$, serves as a representative variable for a variable $R \in M_{\text{var}}$, such that

$$\exists R \in M_{\text{var}} \text{ with } \text{name}(R') = \text{name}(R).$$

- A dedicated class **Prerequisite**, declared as a direct subclass of `owl:Thing` and constrained by $\text{Prerequisite} \sqcap \text{VarClass} \equiv \perp$, is used to specify the patterns that define a prerequisite (i.e., an axiom pattern set). Any variable class D , defined as $D \sqsubseteq \text{Prerequisite}$, can specify a prerequisite if $D \sqsubseteq \neg M_{\text{preq}}:\text{VarClass}$, but $D \sqsubseteq M_{\text{var}}:\text{VarClass}$. Then, the semantics of $M_{\text{preq}}:D$ is inherited from the

Table 3. Example of a model $M1 = M1_{\text{var}} \cup M1_{\text{preq}}$ instantiating the metamodel \mathcal{M} , following the method described in Section 3.1 with as input ontology the AWO [13]. M_{var} and M_{preq} are to specify the variables and prerequisites, respectively; ‘representative’ means that the entity represents a variable in M_{var} .

$M1_{\text{var}}$	$M1_{\text{preq}}$
VarClass \sqsubseteq owl:Thing	VarClass \sqsubseteq owl:Thing {representative}
VarOP \sqsubseteq owl:topObjectProperty	VarOP \sqsubseteq owl:topObjectProperty {representative}
	Prerequisite \sqsubseteq owl:Thing , Prerequisite \sqcap VarClass $\equiv \perp$
Her \sqsubseteq VarClass {OnlyMe}	Her \sqsubseteq VarClass {representative}
Her \equiv AWO:herbivore	A \sqsubseteq VarClass {representative}
A \sqsubseteq VarClass {OnlyMe}	D \sqsubseteq VarClass {representative}
A \equiv AWO:animal	E \sqsubseteq VarClass {representative}
D \sqsubseteq VarClass {MeAndChildren}	Car \sqsubseteq VarClass {representative}
D \equiv AWO:plant	OP2 \sqsubseteq VarOP {representative}
E \sqsubseteq VarClass {MeAndChildren}	OP3 \sqsubseteq VarOP {representative}
E \equiv AWO:plant	OP4 \sqsubseteq VarOP {representative}
Z \sqsubseteq VarClass {MeAndChildren}	B \sqsubseteq Prerequisite
Z \equiv AWO:plant	C \sqsubseteq Prerequisite
B \sqsubseteq VarClass {OnlyChildren}	X \sqsubseteq Prerequisite
C \sqsubseteq VarClass {OnlyChildren}	B $\sqsubseteq \exists$ OP3.Z
X \sqsubseteq VarClass {OnlyChildren}	C $\sqsubseteq \neg(D \sqcap (\exists$ OP4.E))
OP2 \sqsubseteq VarOP {OnlyMe}	X \sqsubseteq Her
OP2 \equiv AWO:eats	X \sqsubseteq A
OP3 \sqsubseteq VarOP {MeAndChildren}	X $\sqsubseteq \forall$ OP2.B
OP3 \equiv AWO:is-part-of	
OP4 \sqsubseteq VarOP {MeAndChildren}	
OP4 \equiv AWO:is-part-of	

corresponding variable class defined in M_{var} ; so $M_{\text{preq}}:D$ is the representative of $M_{\text{var}}:D$ in M_{preq} . This has to be followed because of the disjointness between Prerequisite and VarClass, which avoids incoherence within M_{preq} . Then, the set of valid classes for each D depends on the semantics of D as defined in M_{var} . Each class variable D can then be defined by an unbounded number of patterns constructed using OWL 2 DL Description Logic constructors, and can refer to any variable C' or R' declared in VarClass and VarOP, respectively. The semantics of C' and R' are inherited from the corresponding variables C and R defined in M_{var} , respectively.

We illustrate the metamodel and its usage next.

Example of an instantiated model Let us illustrate an example model, $M1$, shown in Table 3, that is instantiated from the metamodel \mathcal{M} . The OWL ontologies $M1_{\text{var}}$ and $M1_{\text{preq}}$ represent the variables and prerequisites of $M1$, respectively. The variables of $M1$ are A, D, E, Z, B, C, X, OP2, OP3 and OP4 (see Table 3, 1st column), where each of them can be annotated with one of the following tags: *OnlyMe*, *MeAndChildren* or *OnlyChildren*. Then, the prerequisites are specified through the classes B, C and X (see Table 3, 2nd column). In

Table 4. Example of a result obtained from $M1 = M1_{\text{var}} \cup M1_{\text{preq}}$ with AWO [13].

Selected entities
$C \leftarrow \text{AWO:Elephant}$, $D \leftarrow \text{AWO:Grass}$, $\text{OP4} \leftarrow \text{AWO:is-proper-part-of}$, $E \leftarrow \text{AWO:CarnivorousPlant}$, $X \leftarrow \text{AWO:giraffe}$, $\text{OP2} \leftarrow \text{AWO:eats}$, $B \leftarrow \text{AWO:PlantParts}$, $\text{OP3} \leftarrow \text{AWO:is-part-of}$, $Z \leftarrow \text{AWO:plant}$, $\text{Her} \leftarrow \text{AWO:herbivore}$, $A \leftarrow \text{AWO:animal}$
Concrete axioms
$\text{PlantParts} \sqsubseteq \exists \text{is-part-of.plant}$ $\text{Elephant} \sqsubseteq \neg(\text{Grass} \sqcap (\exists \text{is-proper-part-of.CarnivorousPlant}))$ $\text{giraffe} \sqsubseteq \text{herbivore}$ $\text{giraffe} \sqsubseteq \text{animal}$ $\text{giraffe} \sqsubseteq \forall \text{eats.PlantParts}$
Meaning {This is to have an overview of the meaning of these concrete axioms.}
Question: Is a giraffe a herbivore?
Feedback: A giraffe is a herbivore because a giraffe eats a part of plants, which is a proper part of a plant. A giraffe does not eat an elephant, for instance.

Table 3, 2nd column, presenting $M1_{\text{preq}}$, the variable tagged as *mask* must have been defined in $M1_{\text{var}}$, and the semantics of this variable are inherited from the corresponding variable in $M1_{\text{var}}$. In our example, the semantics of the variable are grounded in AWO [13], but they can also be aligned with upper level ontologies, such as BFO [4], and doing so increases the reusability of a developed model.

From $M1$ with AWO, one can obtain a set of concrete axioms as result; see Table 4. The selected entities satisfy the prerequisites defined in $M1_{\text{preq}}$ according to the variables specified in $M1_{\text{var}}$ (see Table 3). The question and feedback (see Table 4) are natural language renderings of the selected axioms; its generation is not the focus of this paper and they are solely included for illustration.

3.2 Evidence supporting the resolution of limitations L1–L5

We now describe how this model addresses limitations listed in Section 2.

(L1) Formalising complex prerequisites in models instantiated from \mathcal{M} via OWL 2 DL. A model is composed of a set of variables and a set of prerequisites. Each variable v within an instantiated model is explicitly typed as either an OWL class ($v \sqsubseteq M_{\text{var}}:\text{VarClass}$) or an OWL object property ($v \sqsubseteq M_{\text{var}}:\text{VarOP}$), ensuring that each variable inherits all the logical characteristics defined by the OWL 2 DL specification. In the case of OWL Classes, any variable v typed as a class is a logical entity capable of being defined through complex class expressions, including qualified cardinality constraints and boolean combinations. Then, in the case of OWL Object Properties, any variable v typed as an object property can be assigned specific property characteristics, such as transitivity and property chains. These characteristics govern the inferential behaviour of the model during reasoning. A prerequisite D , where $D \sqsubseteq M_{\text{preq}}:\text{Prerequisite}$, is formally defined as a defined class within the model, such that D is formulated

as the conjunction of multiple axiom patterns D_i , i.e., $D \sqsubseteq D_1 \sqcap D_2 \sqcap \dots \sqcap D_n$. Because each D_i is a valid OWL 2 DL class expression, it can encapsulate any combination of boolean operators (intersection, union, complement) and property restrictions. Consequently, the prerequisite D is itself a valid OWL 2 DL class expression. Because the variables and prerequisites are valid OWL 2 DL constructs, the framework can formalise complex questions and feedback patterns through OWL 2 DL expressivity constraints. It is the responsibility of the reasoner to handle those complexities (and maintaining decidability).

(L2) Metamodel \mathcal{M} excluding inappropriate questions and feedback within models. In Section 3.1, we mentioned that the variables in the metamodel are defined in $M_{\text{var}}:\text{VarClass}$ (for classes) and $M_{\text{var}}:\text{VarOP}$ (for object properties). To exclude an element V (where $V \in M_{\text{var}}$), which represents either an entity or a variable denoting a set of entities defined by a (possibly complex) OWL 2 DL axiom, the metamodel allows expressing constraints directly at the model-definition stage. If a class D must satisfy a prerequisite in order to generate questions or feedback, then adding the axiom $D \sqcap V \equiv \perp$ is sufficient to guarantee that all classes C , where $C \sqsubseteq V$ are excluded from the output. Therefore, any exclusion condition expressible in OWL 2 DL can be integrated into the model definition while preserving the intended semantics and taking advantage of reasoning service in OWL 2 DL reasoners.

(L3) Metamodel \mathcal{M} incorporating full logical semantics in the processing of variables. In Section 3.1, variables in \mathcal{M} are defined in such a way that each variable can be semantically grounded by mapping it to an entity in an external ontology. Formally, for any variable $D \in M_{\text{var}}$, a mapping $D \equiv C$ may be established, where C is a class in an ontology F . This mapping ensures that the full logical semantics of C , including all axioms, constraints, and inferred consequences, apply to D . Consequently, for any concrete entity $E \in O_1$ such that $E \sqsubseteq D$, the entity E must satisfy the complete logical definition of C . This guarantees that variables are not treated as untyped placeholders, but rather as semantically meaningful constructs whose interpretation is grounded in OWL 2 DL semantics. The metamodel \mathcal{M} therefore preserves and propagates full logical meaning during the processing of variables.

(L4) Metamodel \mathcal{M} allowing an unbounded number of axiom patterns. In Section 3.1, any axiom pattern D_1, D_2, \dots, D_n of a prerequisite can be defined, such that $D_1 \sqsubseteq M_{\text{preq}}:\text{Prerequisite}$, $D_2 \sqsubseteq M_{\text{preq}}:\text{Prerequisite}$, \dots and $D_n \sqsubseteq M_{\text{preq}}:\text{Prerequisite}$. Then, each D_i , where $1 \leq i \leq n$, can be composed as an arbitrarily complex conjunction $D_i \sqsubseteq A_{i1} \sqcap A_{i2} \sqcap \dots \sqcap A_{im}$. Because neither the number of axiom patterns n nor the number of conjuncts m is restricted, any model that is instantiated from the metamodel \mathcal{M} can be defined with an unrestricted number of axiom patterns.

(L5) Metamodel \mathcal{M} enabling scalability and reusability. As illustrated in Figure 1, our framework clearly separates the modelling of prerequisites from the content-selection algorithm, following the architectural principle in [24]. Because the models are defined independently of any specific application scenario, they can be reused across different domains and question-generation contexts.

In addition, since the construction of the models relies on the expressive and decidable OWL 2 DL, no additional constraints are imposed on the complexity of the models, ensuring that the approach remains scalable as new prerequisite patterns and variables are added, without requiring changes to the metamodel.

Thus, in theory, the metamodel \mathcal{M} addresses limitations L1–L5. In practice, the metamodel \mathcal{M} is validated through its implementation, described in Section 4.2. Each model instantiates the metamodel by implementing two OWL files, M_{var} and M_{preq} (Table 2). As illustrated in Table 3, $M1$ instantiates these as $M1_{\text{var}}$ and $M1_{\text{preq}}$. Each model is therefore fully represented using OWL files. This representation ensures that both variables and prerequisites are expressed in OWL, and that the resulting models comply with the OWL 2 DL profile. Compliance is verified using the OWL 2 DL profile checker in Protégé, ensuring that the models are within OWL 2 DL. Consequently, the mechanism guarantees the use of OWL 2 DL and supports the theoretical evidence presented below. Next, we address L6.

4 Determination algorithm

The first aspect of Algorithm 1, related to M_{var} , concerns finding all concrete entities from the variables V that are used in the axiom pattern set (APS), i.e., a prerequisite for generating the question and feedback, that is modelled in M_{preq} . Second, it finds all concrete axioms \mathcal{CA} from APS that use the variables V . To achieve this, we designed a recursive algorithm, ensuring completeness and soundness, while avoiding the need to traverse all possible cases.

The naive way of computing the results is to test all possible combinations of each axiom pattern a_i with all possible entities obtained from $\beta:\text{VarClass}$, $\beta:\text{VarOP}$ used in α , where α is the prerequisite relying on M_{preq} and β denotes the variables used in M_{var} . Because it is a combinatorial problem, the computation grows exponentially. Our analysis led to two key optimisation factors, being:

- **Factor 1:** There are some cases that do not need to be processed. With this problem, if α_i (axiom pattern) results in a concrete axiom c_i , finding c_{i+1} from α_{i+1} is necessary, otherwise we can skip $\alpha_{i+1}, \alpha_{i+2}, \dots, \alpha_p$. Factor 1 holds because if we have an axiom pattern $\alpha = \alpha_1 \sqcap \alpha_2 \sqcap \dots \sqcap \alpha_p$ and $\exists \alpha_i$ such that $O_1 \not\models \alpha_i$ (meaning O_1 does not model one of the axiom patterns), $O_1 \not\models \alpha$ (meaning O_1 does not model the axiom pattern), where α_i is axiom pattern with $1 \leq i \leq p$.
- **Factor 2:** The search space can be reduced by only working with concrete entities that have been validated previously (lines 6, 7, 18 and 19 in Algorithm 1). Consider the axiom pattern $\alpha(V, T)$, where $V = \{v_1, v_2, \dots, v_k\}$ is the set of variables used to define α , and $T = \{T_1, T_2, \dots, T_k\}$ is a set, where $T_i = \{t_1, t_2, \dots, t_l\}$ is a set of entities where v_i can be replaced by t_j , with $1 \leq i \leq k$ and $1 \leq j \leq l$. In line with Factor 1, we can define an axiom pattern $\alpha(V, T)$ as $\alpha(V, T) = \alpha_1(V, T) \sqcap \alpha_2(V, T) \sqcap \dots \sqcap \alpha_p(V, T)$. With Factor 2, we argue that $\alpha(V, T) = \alpha_1(V, T^1) \sqcap \alpha_2(V, T^2) \sqcap \dots \sqcap \alpha_p(V, T^p)$, where $T \supseteq T^1 \supseteq T^2 \supseteq \dots \supseteq T^{p-1} \supseteq T^p$. The proof is as follows. For

Algorithm 1 Determination algorithm

```
1: Input:  $O_1, \alpha, \beta$   $\triangleright O_1$ : input ontology,  $\alpha$ : a model based on  $M_{\text{preq}}$  for a type of
   question with feedback (an APS),  $\beta$ : a model based on  $M_{\text{var}}$  defining all variables
2: Initialisation:  $i \leftarrow 1, p \leftarrow$  number of axiom patterns in  $\alpha, E_e \leftarrow \emptyset, r \leftarrow \emptyset, R \leftarrow \emptyset$ 
    $\triangleright i$ : index of an axiom pattern  $\in \alpha, E_e$ : set of entities that are excluded from the
   research space,  $r$ : partial results,  $R$ : final results
3:  $A \leftarrow \text{GETSUBAXIOMPATTERNS}(\alpha) \triangleright A$ : list of axiom patterns  $A = [a_1, a_2, \dots, a_p]$ 
   belonging to  $\alpha$ 
4: procedure FINDVALIDAXIOMS( $i, A, E_e, r, O_1$ )
5:    $a \leftarrow a_i \in A$   $\triangleright$  where  $1 \leq i \leq p$ 
6:    $T \leftarrow \text{FETCHENTITIESPERVARIABLE}(a, \beta) \triangleright$  obtaining  $T$ , storing all entities for
   each variable used in  $a$  based on  $\beta$ :VarClass and  $\beta$ :VarOP
7:    $T \leftarrow T \setminus E_e$   $\triangleright$  only working with a set of valid entities
8:    $S \leftarrow \text{GETCANDIDATEAXIOMS}(a, T) \triangleright$  where  $S_i = \{s_{i1}, s_{i2}, \dots, s_{in}\}$  a set of all
   possible concrete axioms following the axiom pattern  $a$ , using the set of entities  $T$ 
9:    $i++$   $\triangleright$  working with  $a_{i+1} \in A$  in the next recursion
10:  for all  $s_{ij}$  in  $S_i$  do
11:    if  $O_1 \models s_{ij}$  then
12:       $\text{new}(r')$ 
13:       $r'.\text{parent}(r)$ 
14:       $r'.\text{add}(a, s_{ij})$   $\triangleright$  storing the valid result  $s$  in a leaf
15:      if  $i = p$  then  $\triangleright$  A list of valid concrete patterns is found.
16:         $R.\text{add}(r')$   $\triangleright$  list of concrete axioms from  $a$  can be obtained from  $r'$ 
17:      else
18:         $ex \leftarrow \text{GETEXCLUDEDENTITIES}(s_{ij})$ 
19:         $E_e \leftarrow E_e \cup ex$ 
20:        FINDVALIDAXIOMS( $i, A, E_r, r', O_1$ )  $\triangleright$  redoing the same steps for
    $a_{i+1} \in A$  (if  $i < p$ )
21:      end if
22:    end if
23:  end for
24: end procedure
```

$\alpha_i(V, T^i)$, which results into a concrete axiom $\gamma_i, \exists V_a$, where V_a is replaced by an entity $e \in T_a$, where $T_a = \{t_1, t_2, \dots, t_l\}$ (t_1, t_2, \dots, t_l are entities). At the i^{th} iteration, the set of entities $\Gamma = T_a \setminus \{e\}$ do not satisfy γ_i from $\alpha_i(V, T^i)$. Therefore, $T_a \leftarrow \{e\}$, and $\forall T \in \{T_1^{i+1}, T_2^{i+1}, \dots, T_k^{i+1}\}, T \leftarrow T \setminus \Gamma$ since all entities $\in \Gamma$ do not satisfy $\alpha_i(V, T^i)$, using Γ with $\alpha_{i+1}(V, T^{i+1})$ is a waste of time. This confirms that $T^i \supseteq T^{i+1}$. By recurrence, we obtain $T \supseteq T^1 \supseteq T^2 \supseteq \dots \supseteq T^{p-1} \supseteq T^p$. Therefore, over each iteration, the size of concrete entities decreases, i.e., the search space is reduced for each iteration.

To avoid testing all possible cases, first, we take advantage of Factor 1 and define a recursion algorithm through the procedure FINDVALIDAXIOMS (lines 5-24) that is initialised with the default values that are given in line 2. Each instance of the recursion works with a particular axiom pattern a_i of A (line 6), represented by a along the algorithm, where $1 \leq i \leq p$. The algorithm finds the set of all concrete axioms $S_i = \{s_{i1}, s_{i2}, \dots, s_{in}\}$ that follow the considered axiom pattern a using

only the valid set of entities T , and the procedure checks if ontology O_1 entails each s_{ij} ($O_1 \models s_{ij}$) (lines 10-11). If so, the concrete axiom s_{ij} is stored (lines 12-14). Then, if no more axiom pattern exists, r' , which represents a path of specific valid axioms, is added to the final results R (lines 15-16). Otherwise, Algorithm 1 continues with the next axiom pattern a_{i+1} with new parameters (line 20). If $O_1 \not\models s_{ij}$, Algorithm 1 ignores s_{ij} , avoiding unnecessary steps following Factor 1.

We now describe the different procedures used in Algorithm 1. In line 3, it uses the procedure `GETSUBAXIOMPATTERNS(α)` to obtain the list of axiom patterns A of α . The procedure `FETCHENTITIESPERVARIABLE(a, β)`, in line 6, consists of retrieving all concrete entities that can replace each variable used in a , and it stores the results in T . Consider that the axiom pattern a is defined as follow, $a = \langle V, \mathcal{A}(V) \rangle$, where $V = \{v_1, v_2, \dots, v_n\}$ is the set of variables, and $\mathcal{A}(V)$ is a logical expression over V , where n is the number of variables. The result T from `FETCHENTITIESPERVARIABLE` is as follows, $T = \{t_1, t_2, \dots, t_n\}$, where $t_i = \{\forall x \in O_1 \mid x \sqsubseteq v_i\}$, where $v_i \sqsubset \beta:\text{VarClass}$ or $v_i \sqsubset \beta:\text{VarOP}$, and $1 \leq i \leq n$. The procedure `GETCANDIDATEAXIOMS(a, T)` that is used in line 8 consists of retrieving all concrete axioms following the axiom pattern a by only using the validated entities in T to replace the variables used in a . At this stage, the procedure does not yet take into account if $O_1 \models a$ or not. In line 18, following Factor 2, `GETEXCLUDEDENTITIES(s_{ij})` consists of fetching the set of entities that are excluded $ex = ex_1 \cup ex_2 \cup \dots \cup ex_m$, such that for each entity e_k used in s_{ij} that replaces the variable v_k used in a , we have $ex_k = \{\forall x \mid x \sqsubseteq v_k\} \setminus \{e_k\}$, where m is the number of variables defining a and $1 \leq k \leq m$.

We also used OWL API [12] and Hermit [11] to implement Algorithm 1. Sections 4.1 and 4.2 present the analysis of Algorithm 1, addressing L6.

4.1 Theoretical analysis

Complexity analysis. Our theoretical analysis shows that the base case and the recursive case are nondeterministic. $O_1 \not\models s_{ij}$ and $O_1 \models s_{ij}$ depends on the input ontology O_1 , the axiom pattern α ; and they affect T that is derived from β (lines 6-7), and S (line 8). Therefore, the path representing the recursive algorithm is also nondeterministic. Nonetheless, our analysis shows that the upper and lower bound complexities of Algorithm 1 are $\mathcal{O}(n^p)$ and $\Omega(n)$, respectively, where p is the number of axiom patterns per prerequisite, and $n = \max(|S_1|, |S_2|, \dots, |S_p|)$ (see line 8). We can pick the maximum number because our interest is in the bound. Therefore, an empirical study to understand the algorithm's behaviour using real ontologies is presented in Section 4.2. The complete analysis can be found in the supplementary material.

Completeness and soundness. Since Algorithm 1 makes use of a Hermit reasoner [11] for computing $O_1 \models s_{ij}$, which is a complete algorithm, and we proved that optimisation factors Factor 1 and Factor 2 do not exclude any valid solutions, Algorithm 1 is complete.

An algorithm is sound if its results are valid and correct. Algorithm 1 cannot produce incorrect consequences when evaluating entailments ($O_1 \models s_{ij}$), because we use the Hermit reasoner [11] for that, and the Hermit reasoner is proven to

Table 5. Main ontology metrics of AWO and EXMO.

ontology	$ axioms $	$ classes $	$ sub-classes $	$ disjoint\ classes $	$ OPs $	$ sub-OPs $
AWO	101	56	73	8	48	1
EXMO	855	407	516	28	117	119

be sound. Nonetheless, the outputs of Algorithm 1 may not be meaningful if the ontologies in input O_1 and the OWL 2 DL-based models are not well-formed.

4.2 Empirical analysis

The complexity varies depending on the ontology and the input model. Therefore, the empirical analysis aims to determine the difference between the number of all possible cases corresponding to the theoretical complexity $\mathcal{O}(n^p)$ and the number of calls to $O_1 \models s_{ij}$ (line 11) in real cases.

Materials and Methods. We selected two ontologies: African Wildlife Ontology (AWO) [13] and Exercise Medicine Ontology (EXMO) [19]. AWO was chosen to define complex models intended to stress-test the algorithm, and EXMO because it reuses BFO [4], to demonstrate that our model overcomes limitation L3. We developed 10 and 6 prerequisites (α) for AWO and EXMO, respectively, where the number of axiom patterns (β) varied between 1 and 9. Because our interest is the *speed-up* that is the ratio between the number of all possible cases and the number of calls of $O_1 \models s_{ij}$, the prerequisites that we developed are intended to stress Algorithm 1 by developing complex models. For example, as shown in Table 6, p is high for most of the models (the number of axiom patterns defining a prerequisite).

We conducted the experiments with a laptop with an i7-7700HQ CPU at 2.80 GHz (8 CPUs) and 16 GB of memory.

Results. The results are presented in Table 6. The most important columns are \mathcal{C} and \mathcal{N} , which are the number of calls of $O_1 \models s_{ij}$ in Algorithm 1 (line 11) and the number of all possible cases corresponding to the theoretical complexity $\mathcal{O}(n^p)$ (brute force), respectively. *speed-up*, the ratio \mathcal{N}/\mathcal{C} , enables us to determine the possibility of generating questions and feedback from an instance of the OWL 2 DL-based model. In this paper, *speed-up* is based on the number of steps. From our experiment, there are 3 cases:

1. *speed-up* > 1: This concerns most of the cases (e.g., `awo3` and `exmo4`). This means that the computation is substantially lower than the upper bound $\mathcal{O}(n^p)$ in the case where $p > 1$. This happens because in real scenarios, $O_1 \not\models s_{ij}$ is satisfied often, preventing the algorithm from traversing a lot of unnecessary paths within the recursion tree thanks to the Factor 1 and Factor 2 (See details in the description of the algorithm in the paper.).
2. *speed-up* = 1: This is the case where $p = 1$ (`awo7`, `awo9`), there is no gain because we have $\mathcal{O}(n) = \Omega(n) = \Theta(n)$.
3. *speed-up* < 1: There is a bad design of the question and feedback prerequisites: `exmo2` and `exmo5`. They should have been expressed with $p = 1$.

Table 6. Results of the empirical experiments. *A*: questions and feedback prerequisites (axiom patterns); *awo_k* for $1 \leq k \leq 10$ are prerequisites for AWO, and likewise for EXMO; *p*: number of axiom patterns for each prerequisite; *b*: number of variables in a prerequisite; *q + f*: number of generated questions and feedback; *C*: number of calls of $O_1 \models s_{ij}$ in Algorithm 1 (line 11); *N*: number of all possible cases (complexity $\mathcal{O}(n^p)$ (brute force)); *t(s)*: duration of the generation by using Algorithm 1; *speed-up*, i.e., N/C shows the ratio between computing all cases and the strategy of Algorithm 1. *awo₃* is the model presented in *M1* (Section 3.1).

<i>ontology</i>	<i>A</i>	<i>p</i>	<i>b</i>	<i>q + f</i>	<i>C</i>	<i>N</i>	<i>speed-up</i>	<i>t(s)</i>
AWO	<i>awo₁</i>	3	8	44	167	124416	745,01	1,055
	<i>awo₂</i>	4	9	1920	6227	32099328	5154,86	19,262
	<i>awo₃</i>	5	11	1920	13727	1.11×10^{11}	8058046,00	41,258
	<i>awo₄</i>	4	9	2	26	41472	1595,08	0,009
	<i>awo₅</i>	4	13	104	24360	5.05×10^8	20715,57	99,933
	<i>awo₆</i>	2	3	11	3612	3111696	861,49	0,145
	<i>awo₇</i>	1	3	68	31752	31752	1	48,406
	<i>awo₈</i>	2	4	93	36792	56010528	1522,36	65,925
	<i>awo₉</i>	1	3	145	31752	31752	1	58,123
	<i>awo₁₀</i>	2	4	68	66360	5.04×10^8	7596,37	186,681
EXMO	<i>exmo₁</i>	8	15	30	13656	37015056	2710,53	3,933
	<i>exmo₂</i>	7	15	202	20292	20280	0,999409	4579,450
	<i>exmo₃</i>	8	16	340	37400	1.23×10^8	3299,03	7457,569
	<i>exmo₄</i>	9	18	66	25980	4.81×10^9	185217,80	8290,063
	<i>exmo₅</i>	7	15	202	20292	20280	0,999409	4493,318
	<i>exmo₆</i>	8	16	0	20292	2.06×10^8	10134,00	7079,759

From the results, the time $t(s)$ recorded for finding the concrete axioms is relatively acceptable in real scenarios, taking into account that for each call, there are β operations to compute the entailment $O_1 \models s_{ij}$ by means of Hermit reasoner [11]. We are now justifying that Algorithm 1 is complete and sound, guaranteeing the accuracy and the coverage of the outputs.

The results show that most of the time, *speed-up* is very high, meaning the computation is not close to the upper bound complexity in most cases with real ontologies, and we obtain an average *speed-up* of 518600.13. Then, the average time to find all axioms for each prerequisite is 33.78 minutes. There is no *speed-up* if $p = 1$. This is as expected because we have $\mathcal{O}(n) = \Omega(n) = \Theta(n)$ in this case. Then, the prerequisite is not well-formed if $p < 1$.

In summary, through both theoretical and empirical analysis, we demonstrate that Algorithm 1 effectively addresses L6.

5 Discussion

The new model for representing types of questions and feedback using OWL 2 DL and accompanied by the optimised algorithm, resolves limitations L1-L6. This

Table 7. Comparison with comparable studies. ‘-’: the authors did not report whether the pattern logic representing the types of questions and feedback is embedded in the algorithm or not; ‘B/AP’: the model is based on axiom patterns; ‘question card’ is based on non-standard XML; ‘*’: partially solved; empty cell: the problem is not solved.

Study	Modelling	L1	L2	L3	L4	L5	L6
Abacha et al. [1]	- (B/AP)					-	✓
Vinu et al. [31]	- (B/AP)					-	✓
Venugopal et al. [30]	- (B/AP)					-	✓
Leo et al. [18]	- (B/AP)					-	✓
Mahlaza et al. [22]	- (B/AP)					-	✓
Raboanary et al. [25,26]	Algorithm (B/AP)						✓
Demaidi et al. [8]	Algorithm (B/AP)						✓
Raboanary and Keet [24]	Question card (B/AP)		*		✓	✓	✓
Bühmann et al. [5]	SPARQL			✓	✓		✓
This work	OWL 2 DL-based	✓	✓	✓	✓	✓	✓

enables the community to develop complex prerequisites for generating complex questions and feedback from ontologies. To the best of our knowledge, only [25,26] integrated DOLCE categories in some of their prerequisites, whereas, thanks to this novel framework, one could use one’s foundational ontology of choice, which can require more expressivity and reasoning capabilities. One can also generate educational quizzes in the spirit of [28]. In this case, complex questions and feedback can be created by only developing models in OWL 2 DL, and then use Algorithm 1 to find the concrete axioms, with only the verbalisation remaining. Furthermore, this framework can also be used for retrofitting competency questions on ontologies as done in [2]. However, unlike their LLM-based approach, our method can be grounded directly in axiom patterns.

The quality of the final rendering of the output in natural language is important in Natural Language Generation [29,21,27]. As presented in Fig. 1, from the outputs of Algorithm 1, any realiser of choice can be used for the final rendering.

Table 7 includes the comparison of the framework to relevant existing studies. As can be seen, the existing approaches do cover parts of the capability of our framework. Among them, Raboanary and Keet [24] address L4, L5, L6, and partially L2 because the filtering mechanism does not exploit the OWL 2 DL language (it relies on an algorithmic approach, even though the definition is embedded in the model). This is still close to what Leo et al. [18] did to some extent. Furthermore, although [24] describes a content-selection algorithm within their architecture, it is not demonstrated whether that algorithm is optimal. Also, it cannot support the new OWL 2 DL-based models, as it operates on a specific representation called a question card, with which limitations L1, L3, and the full extent of L2 cannot be addressed.

The instantiation of models from the metamodel is guided. Designers are expected to reuse the metamodel files as a basis and follow the specifications described in Section 3.1. In addition, model development in Protégé enables syntax verification, class satisfiability checking, and reasoning to assess the models. As

shown in the results, performance depends on the design of the prerequisites, as the recursive algorithm exhibits non-deterministic behaviour. However, based on the optimisation factors of the algorithm (Factors 1 and 2 in Section 4), several guidelines have emerged for constructing an effective prerequisite. For instance, sharing as many variables as possible between axiom patterns helps reduce the search space at each step (Factor 2). Furthermore, axiom patterns should preferably be expressed in their shortest possible form, while using a larger number of axiom patterns can better leverage Factor 1. In addition, arranging axiom patterns from less expressive to more expressive can improve the performance of Algorithm 1 by optimising the execution of the instruction at line 11.

6 Conclusions

We proposed a framework composed of a metamodel in OWL 2 DL with which axiom pattern sets can be written, and an optimised content selection algorithm to extract the relevant axioms from an ontology. The theoretical assessment, implementation, and evaluation demonstrated that the models instantiated from the metamodel can represent types of questions and feedback that use OWL 2 DL constructors, and that it is possible to find the concrete axioms from the OWL 2 DL-based model efficiently by means of a complete and sound algorithm. This new framework can enhance question and feedback generation from ontologies; notably, the generation of questions and feedback, but also other types of questions, such as retrofitting competency questions on ontologies or for education or pop quiz activities.

For future work, we aim to develop a tool that supports model creation from the metamodel, and to extend this framework for better retrofitting competency questions and to enable even more complex pattern structures.

Supplementary Material Statement: The supplementary materials are available at <https://github.com/toky-raboanary/FrameworkQFgen>, which contains the prerequisites (APS) that we developed to obtain the results (see Table 6), the implemented Algorithm 1, the ontologies used in the experiments, the outputs, the summary of the results, a document with the details of the theoretical analysis of Algorithm 1, and examples of axiom patterns. The different folders are described in its `readme.txt` file. We used OWL API v5.1.8 and Hermit reasoner v1.4.3.517, as declared in the `pom.xml`, together with JAVA 17 SDK. The project can be opened with IntelliJ IDEA 2023.2 (Community Edition). The models and the metamodel can be edited with Protégé v5.5.0 (or later).

Generative AI Statement: One of the authors used Grammarly for grammar checking and the free version of ChatGPT to help find appropriate wording.

Acknowledgments: This work is based on the research supported in part by the National Research Foundation of South Africa (no.: SAI240823262612). This work was financially supported by Hasso Plattner Institute for Digital Engineering through the HPI Research School at UCT.

References

1. Abacha, A.B., Dos Reis, J.C., Mrabet, Y., Pruski, C., Da Silveira, M.: Towards natural language question generation for the validation of ontologies and mappings. *Journal of Biomedical Semantics* **7**(1), 1–15 (2016)
2. Alharbi, R., Tamma, V., Grasso, F., Payne, T.: An experiment in retrofitting competency questions for existing ontologies. In: *Proceedings of the 39th ACM/SI-GAPP Symposium on Applied Computing*. pp. 1650–1658 (2024)
3. Arenas, M., Gottlob, G., Pieris, A.: Expressive languages for querying the semantic web. *ACM Transactions on Database Systems* **43**, 13 (2018)
4. Arp, R., Smith, B., Spear, A.D.: *Building ontologies with basic formal ontology*. Mit Press (2015)
5. Bühmann, L., Usbeck, R., Ngomo, A.C.N.: ASSESS—automatic self-assessment using linked data. In: Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d’Aquino, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Thirunarayan, K., Staab, S. (eds.) *Proceedings (Part II) of the 14th International Semantic Web Conference, ISWC 2015*. LNCS, vol. 9367, pp. 76–89. Springer, Bethlehem, PA, USA (2015)
6. Chaudhri, V.K., Clark, P.E., Overholtzer, A., Spaulding, A.: Question generation from a knowledge base. In: Janowicz, K., Schlobach, S., Lambrix, P., Hyvönen, E. (eds.) *Proceedings of the 19th International Conference on Knowledge Engineering and Knowledge Management, EKAW 2014*. LNAI, vol. 8876, pp. 54–65. Springer, Linköping, Sweden (2014)
7. Chaudhri, V., Cheng, B., Overholtzer, A., Roschelle, J., Spaulding, A., Clark, P., Greaves, M., Gunning, D.: *Inquire Biology: A textbook that answers questions*. *AI Magazine* **34**(3), 55–72 (2013)
8. Demaidi, M.N., Gaber, M.M., Filer, N.: OntoPeFeGe: Ontology-based personalized feedback generator. *IEEE Access* **6**, 31644–31664 (2018)
9. Dong, C., Li, Y., Gong, H., Chen, M., Li, J., Shen, Y., Yang, M.: A survey of natural language generation. *ACM Computing Surveys* **55**(8), 1–38 (2022)
10. Gatt, A., Reiter, E.: SimpleNLG: A realisation engine for practical applications. In: Krahmer, E., Theune, M. (eds.) *Proceedings of the 12th European Workshop on Natural Language Generation, ENLG 2009*. pp. 90–93. Association for Computational Linguistics (ACL), Athens, Greece (Mar 2009)
11. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An OWL 2 reasoner. *Journal of Automated Reasoning* **53**(3), 245–269 (2014)
12. Horridge, M., Bechhofer, S.: The OWL API: A java API for OWL ontologies. *Semantic Web* **2**(1), 11–21 (2011)
13. Keet, C.M.: The African Wildlife Ontology tutorial ontologies. *Journal of Biomedical Semantics* **11**(4), 1–11 (2020)
14. Kumar, A.P., Nayak, A., K, M.S., Chaitanya, Ghosh, K.: A novel framework for the generation of multiple choice question stems using semantic and machine-learning techniques. *International Journal of Artificial Intelligence in Education* **34**(2), 332–375 (2024)
15. Kurdi, G., Leo, J., Parsia, B., Sattler, U., Al-Emari, S.: A systematic review of automatic question generation for educational purposes. *International Journal of Artificial Intelligence in Education* **30**(1), 121–204 (2020)
16. Kusuma, S.F., Siahaan, D.O., Fatchah, C.: Automatic question generation in education domain based on ontology. In: *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*. pp. 251–256. IEEE (2020)

17. Kusuma, S.F., Siahaan, D.O., Faticah, C.: Automatic question generation with various difficulty levels based on knowledge ontology using a query template. *Knowledge-Based Systems* **249**, 108906 (2022)
18. Leo, J., Kurdi, G., Matentzoglou, N., Parsia, B., Sattler, U., Forge, S., Donato, G., Dowling, W.: Ontology-based generation of medical, multi-term MCQs. *International Journal of Artificial Intelligence in Education* **29**(2), 145–188 (2019)
19. Liu, X., Yang, Y., Zong, H., Zhang, K., Jiang, M., Yu, C., Chen, Y., Bao, T., Li, D., Wang, J., et al.: Core reference ontology for individualized exercise prescription. *Scientific Data* **11**(1), 1349 (2024)
20. Mahlaza, Z., Keet, C.M.: Surface realisation architecture for low-resourced African languages. *ACM Transactions on Asian and Low-Resource Language Information Processing* **22**(3), 1–26 (2023)
21. Mahlaza, Z., Keet, C.M.: A classification of grammar-infused templates for ontology and model verbalisation. In: Garoufallou, E., Fallucchi, F., Luca, E.W.D. (eds.) *Proc. of MTSR'19. CCIS*, vol. 1057, pp. 64–76. Springer, Rome, Italy (2019)
22. Mahlaza, Z., Keet, C.M.: OWLSIZ: An isiZulu CNL for structured knowledge validation. In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. pp. 15–25 (2020)
23. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: Ontology library. WonderWeb Deliverable D18 (ver. 1.0, 31-12-2003). (2003), <http://wonderweb.semanticweb.org>
24. Raboanary, T., Keet, C.M.: An architecture for generating questions, answers, and feedback from ontologies. In: Garoufallou, E., Vlachidis, A. (eds.) *Proc of MTSR'22. CCIS*, Springer, London, UK (2023)
25. Raboanary, T., Wang, S., Keet, C.M.: Generating answerable questions from ontologies for educational exercises. In: Garoufallou, E., Ovalle-Perandones, M.A., Vlachidis, A. (eds.) *Proc of MTSR'21. CCIS*, vol. 1537, pp. 28–40. Springer (2022)
26. Raboanary, T.H., Wang, S., Keet, C.M.: Towards the generalisation of the generation of answerable questions from ontologies for education. *International Journal of Metadata, Semantics and Ontologies* **16**(1), 86–103 (2022)
27. Reiter, E.: *Natural Language Generation*. Springer Nature (2024)
28. Rodríguez Rocha, O., Faron Zucker, C.: Automatic generation of quizzes from DBpedia according to educational standards. In: Lahoud, I., Cardoso, E., Matta, N. (eds.) *Proceedings fo the 3rd Educational Knowledge Management Workshop, EKM 2018*. pp. 1035–1041. Lyon, France (Apr 2018), april 23 - 27, 2018
29. Tang, C., Guerin, F., Li, Y., Lin, C.: Recent advances in neural text generation: A task-agnostic survey. *arXiv.org abs/2203.03047* (2022)
30. Venugopal, V.E., Kumar, P.S.: Automated generation of assessment tests from domain ontologies. *Semantic Web* **8**(6), 1023–1047 (2017)
31. Vinu, E., et al.: A novel approach to generate MCQs from domain ontology: Considering DL semantics and open-world assumption. *Journal of Web Semantics* **34**, 40–54 (2015)