Towards a Knowledge-Aware AI
A. Dimou et al. (Eds.)

doi:10.3233/SSW220014

Automatically Drafting Ontologies from Competency Questions with FrODO

Aldo GANGEMI [a,b], Anna Sofia LIPPOLIS [a], Giorgia LODI [a], Andrea Giovanni NUZZOLESE [a,1,2]

[a] *Institute of Cognitive Sciences and Technologies, Italian National Research Council (ISTC-CNR), Via San Martino della Battaglia 44, 00184, Rome, Italy*
[b] *Department of Classical and Italian Philology, University of Bologna, Via Zamboni 32, 40126, Bologna, Italy*

Abstract. We present the Frame-based ontology Design Outlet (FrODO), a novel method and tool for drafting ontologies from competency questions automatically. Competency questions are expressed as natural language and are a common solution for representing requirements in a number of agile ontology engineering methodologies, such as the eXtreme Design (XD) or SAMOD. FrODO builds on top of FRED. In fact, it leverages the frame semantics for drawing domain-relevant boundaries around the RDF produced by FRED from a competency question, thus drafting domain ontologies. We carried out a user-based study for assessing FrODO in supporting engineers for ontology design tasks. The study shows that FrODO is effective in this and the resulting ontology drafts are qualitative.

Keywords. Ontology Engineering, Ontology, Machine Reading, Knowledge graph, Knowledge representation

1. Introduction

Competency questions [1] (CQs) expressed as natural language are a common solution for representing requirements in a number of agile ontology engineering methodologies, such as the eXtreme Design [2] (XD) or SAMOD [3]. In such methodologies most effort lies in the design of ontology modules able to address the CQs that have been previously identified, which is a fully manual activity. Hence, it represents a clear bottleneck for agile methodologies. This is fairly evident in situations in which ontology drafts need to be shared among ontology engineers and domain experts or stakeholders for incremental refinements and/or knowledge exchange. Accordingly, it is utmost important that those ontology drafts, although tentative, must comply with best design practices, e.g. Ontology Design Patterns, and properties, e.g. cognitive ergonomics, transparency and flexibility. In this paper we present the Frame-based Ontology Design Outlet (FrODO), which is a novel method and Web tool for automatically drafting OWL ontologies from CQs. FrODO builds on and benefits from FRED [4] for *machine reading* [5] aimed at gathering RDF from natural language. FRED is a formal machine reader that produces

[1] Corresponding Author: Andrea Giovanni Nuzzolese; E-mail: andreagiovanni.nuzzolese@cnr.it
[2] The authors are sorted alphabetically as they equally contributed to this paper.

RDF graphs from text, which are (i) domain- and task-independent, and (ii) designed according to the frame semantics [6] and ontology design patterns [7]. Hence, FrODO extends FRED specifically on the case of CQs by tailoring the RDF produced by FRED into domain ontologies by leveraging its formal representation based on the frame semantics. The domain ontologies produced by FrODO are drafts that can be used to feed agile ontology design methodologies. Accordingly, in this work we investigate the following research questions:

- *RQ1*: Is frame semantics fair to be exploited for generating well structured ontology drafts from CQs?
- *RQ2*: Do ontology engineers benefit from ontology drafts that are automatically produced from CQs in their ontology engineering tasks?

In order to address the aforementioned research questions we carried out a user-based study. The participants to the study were asked to use FrODO during the design starting from a set of given CQs. The quality of the resulting ontologies was measured with well known structural metrics. The effectiveness of FrODO was measured in terms of usability within the context of a an ontology engineering workflow. The rest of the paper is organised as it follows: Section 2 surveys some related works; Section 3 describes the methodology implemented by FrODO; Section 4 describes the evaluation, presents and discusses the results; finally, Section 5 presents our conclusions and future works.

2. Related Work

Competency questions (CQs) are questions in natural language that define and constrain the scope of knowledge represented in an ontology. As such, they are used in ontology engineering as requirements useful to evaluate an ontology based on its ability to answer each question, particularly in several agile methodologies, such as, the TOVE enterprise modeling approach [8], eXtreme Design (XD) in [2], SAMOD [3], On-To-Knowledge [9]). All the aforementioned methodologies can be defined as test driven [10,11]. Namely, they assess the commitment of ontologies to the requirements by converting CQs into queries, e.g. SPARQL, DL queries, etc. For example, [12] is a Web application designed for providing the XD methodology with a testing toolbox based on the representation of CQs to SPARQL queries. In this context [13] provides a solution for generating SPARQL queries from CQs. Our solution is meant to be used as a component of the aforementioned agile methodologies. However, it does not automatise the generation of queries for testing purposes, but provides a solution for drafting an ontology from its associated CQs automatically. The latter point can be seen an Ontology Learning and Population (OL&P) task [14,15]. Examples of such methods include [16,17,18]. Most of these solutions are implemented on top of machine learning methods. Hence, they are typically data hungry, i.e. they require large corpora, sometimes manually annotated, in order to learn rules for ontology automatic construction. Such rules are defined through a training phase that can take a long time. On the contrary, our solution does not depend on any training and is unsupervised. Other approaches to OL&P use either lexico-syntactic patterns [19], or hybrid lexical-logical techniques [20]. However, to the best of our knowledge no practical tools have emerged so far for doing it automatically while preserving high quality of results. Finally [4] is a formal machine reader able to

transform natural language text into domain-independent formal structured knowledge represented as RDF/OWL. Our solution build on top of FRED for generating domain-dependent ontologies from CQs.

3. The Frame-based ontology Design Outlet

We generate ontologies from CQs by refactoring the RDF graph produced by FRED. This is done by means of graph traversal strategies that exploit the frame semantics [6]. We assume that frames and frame arguments, as represented by FRED, are the key tools to leverage on for drawing domain-relevant boundaries around the classes and properties produced by FRED. This enables the generation of domain ontology drafts. In fact, on one hand frames convey general meaning, i.e. they are bound in FRED to VerbNet frames that are broader and domain-independent concepts. On the other hand, frame arguments enable a solution for specialising such concepts in a specific domain with peculiar knowledge gathered from text directly. Furthermore, frame roles (e.g. agent, patient, theme, etc), that link a frame to its arguments, can be used for introducing peculiar naming conventions, annotations, and axioms in the draft ontologies that are fine-grained to a domain. Frames are represented by FRED in two alternative ways, that is, (i) as *n*-ary relations and (ii) periphrastic relations. Figure 1 depicts the methodology implemented by FrODO through the UML notation for activity diagrams. In such a figure it is fairly evident how FrODO extends FRED (cf. activity 1) by adding frame recognition (cf. activities 2 and 3) and domain ontology generation and enrichment (cf. activities 4 and 5). The activities if Figure 1 are detailed in the subsequent sections.

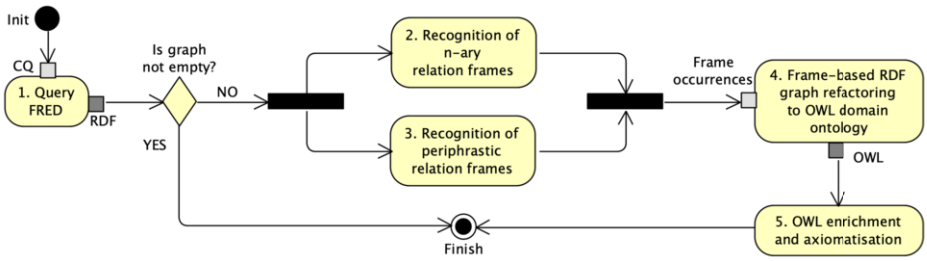


Figure 1. Methodology implemented by FrODO.

3.1. Drafting ontologies from n-ary relations

In the case of *n*-ary relations, a frame occurrence is represented by an individual of a class, which is, in turn, a sub-class of `dul:Event`. The sub-classing of `dul:Event` might be not direct. Hence, we need to traverse the `rdfs:subClassOf` axioms transitively. This can be exemplified with the RDF graph produced by FRED for the CQ *"Who commissioned a component of a system?"*, which is depicted in Figure 2. In such a Figure a frame occurrence based on the *n*-ary pattern is identified by the individual `fred:commission_1`. This individual is a valid frame occurrence as Equation 1 is matched. In fact, `fred:commission_1` is an instance of `fred:Commission`, which is,

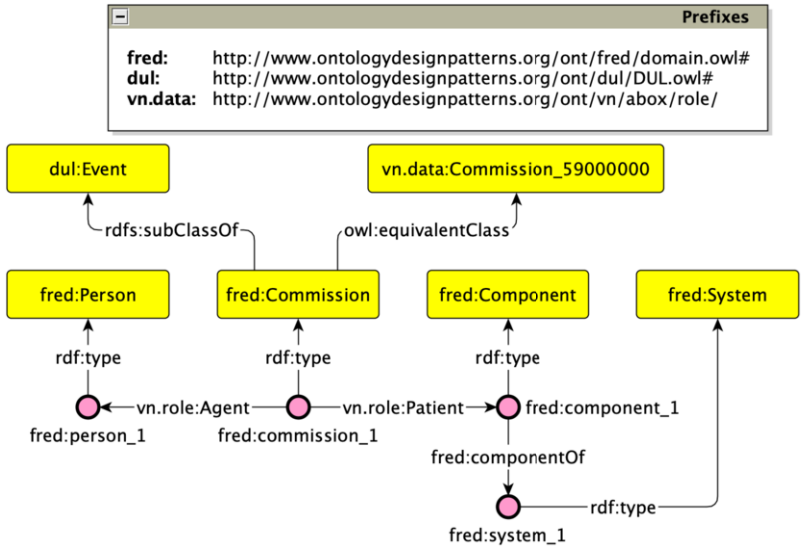


Figure 2. RDF graph produced by FRED for the text *"Who commissioned a component of a system?"*. The graph is drawn with the Graffoo notation [21]

in turn, a sub-class of `dul:Event`. The RDF graph depicted in Figure 2 and the frame identified are possible outputs for the activities 1 and 2 in Figure 1, respectively.

$$F_{n\text{-}ary} \equiv \{f \mid f \in C \wedge C \in \texttt{owl:Class} \wedge C \subseteq^{+} \texttt{dul:Event}\} \tag{1}$$

The arguments of an *n*-ary relation are utmost important for providing the context to an identified frame. In fact, the arguments are the individuals participating in an event, i.e. the frame represented as an *n*-ary relation. These arguments are linked to the event by means of binary predicates, i.e. the frame roles. According to the frame semantics based on VerbNet implemented by FRED possible roles are:

- agentive roles, such as `vn.role:Agent`, `vn.role:Actor`, etc.;
- passive roles, such as `vn.role:Patient`, `vn.role:Patient1`, etc.;
- thematic roles, such as `vn.role:Theme`, `vn.role:Theme1`, `vn.role:Experiencer`;
- oblique roles, such as `vn.role:Asset`, `vn.role:Time`, etc.;
- roles based on periphrastic relations, such as `fred:at`, `fred;composeOf`, `fred:personOf`, etc..

Domain classes in a draft ontology are defined by constructing compound terms from frame occurrences represented with the *n*-ary pattern and their arguments. In this construction the arguments with a passive role are core. In fact, FrODO retrieves the name of the class typing the argument of an *n*-ary that plays a passive role. This class name is then concatenated with the name of the class of the frame occurrence (i.e. the target *n*-ary) itself. The latter name is first declined into gerund form. FrODO does not take into account agentive roles for this construction as they are typically bound to individuals that are typed with classes that are too broader or not constrained to a specific domain, e.g. `fred:Agent`, `fred:Person`, etc. This is due to the fact that

the arguments with agentive roles are typically produced from pronouns, e.g. *"Who"*, *"What"*, *"Which"*, etc., by following a shared pattern for defining CQs [22]. On the contrary, arguments with a passive role are typically associated with domain-relevant terms in CQs, e.g. *"component"*. As an example, in Figure 2 the type of the frame occurrence is `fred:Commission`, while the type of its argument playing a passive role is `fred:Component`. Hence, the new class generated by FrODO for representing a domain-relevant *n*-ary in the draft ontology is `:ComponentCommissioning`[3]. The second step of the approach is to define the classes and properties to use as the arguments of the new *n*-ary classes introduced in a draft. This is done by reusing the classes typing the arguments of a frame occurrence in FRED regardless to the specific role played in the *n*-ary relation. Based on our running example, this means that both `:Component` and `:Person` are defined as classes in the ontology drafted by FrODO. Then, those classes are linked to the *n*-ary they belong to by defining new object properties in place of the frame roles provided by FRED. These object properties are constructed by applying a naming convention based on the template `:involves{ClassName}`, where `{ClassName}` is replaced by the actual name of the class being an argument of the target *n*-ary with a camel case notation. For example, the object properties `:involvesComponent` and `:involvesPerson` are generated for the classes `:Component` and `:Person`, respectively. We opt for the term *"involves"* as it evokes the involvement of a participant, i.e. the argument, into a situation, i.e. the frame. The generated object properties are then provided with domain and range axioms. The range axioms are set to the corresponding argument of the *n*-ary, e.g. `:Component` and `:Person` are the range of `:involvesComponent` and `:involvesPerson`, respectively. On the contrary, domain axioms are set to `owl:Thing`, hence they are kept open. The association with the corresponding *n*-ary relation is materialised by means of existential restriction axioms defined locally to the class representing the *n*-ary relation following the pattern defined in Equation 2. This Equation allows FrODO to cope with the activity 2 of Figure 1.

$$C_{n\text{-}ary} \sqsubseteq \exists r_{arg}.C_{arg}, \qquad n\text{-}ary \in F_{n\text{-}ary}, \forall \langle p, arg \rangle \text{ s.t. } p(n\text{-}ary, arg) \in G_{FRED} \tag{2}$$

In Equation 2 (i) $C_{n\text{-}ary}$ is the class generated by FrODO from the *n*-ary frame occurrence produced by FRED, which belongs to $F_{n\text{-}ary}$; (ii) C_{arg} is the class generated by FrODO for an argument of the *n*-ary in FRED; (iii) r_{arg} is the object property generated by FrODO for an argument of the *n*-ary in FRED; (iv) p is a frame role, i.e. an RDF predicate, that links the *n*-ary frame occurrence and the corresponding argument in the graph produced FRED, which is G_{FRED}. For each generated object property its corresponding inverse is materialised. The naming convention for inverse object properties is based on the template `:is{ClassName}InvolvedIn`. In this template `{ClassName}` is substituted with the actual name of the class being an argument of the target *n*-ary with a camel case notation. For example, `:isComponentInvolvedIn` and `:isPersonInvolvedIn` are defined as inverse object properties of `:involvesComponent` and `:involvesPerson`, respectively. Addiontally, for each generated class a taxonomy is inferred following the compositional semantics. For example, `:ComponentCommissioning` is declared to be `rdfs:subClassOf :Commissioning`. Finally, each class and property defined in the draft ontology is as-

[3] In this paper the prefix `:` is reserved for referring to the namespace used by FrODO.

sociated with a human readable label, i.e. `rdfs:label`. The following code is the OWL produced by FrODO for the CQ used so far in our running example. The OWL is serialised with the Manchester syntax. All the solutions explained for generating axioms and labels are part of the activity 5 in Figure 1.

```
ObjectProperty: involvesComponent
     Annotations: rdfs:label "involves component"
     Domain: owl:Thing
     Range: Component
     InverseOf: isComponentInvolvedIn
ObjectProperty: involvesPerson
    Annotations: rdfs:label "involves person"
    Domain: owl:Thing
    Range: Person
    InverseOf: isPersonInvolvedIn
ObjectProperty: isComponentInvolvedIn
    Annotations rdfs:label "is component involved in"
    Domain: Component
    Range: owl:Thing
    InverseOf: involvesComponent
ObjectProperty: isPersonInvolvedIn
    Annotations: rdfs:label "is person involved in"
    Domain: Person
    Range: owl:Thing
    InverseOf: involvesPerson
Class: Commissioning
    Annotations: rdfs:label "Commissioning"@en
Class: Component
    Annotations: rdfs:label "Component"
Class: ComponentCommissioning
    Annotations: rdfs:label "Component commissioning"@en
    SubClassOf: Commissioning,
                involvesComponent some Component,
                involvesPerson some Person
Class: Person
    Annotations: rdfs:label "Person"
```

3.2. *Drafting ontologies from periphrastic relations*

Periphrastic relations occur in FRED when it generates meaningful object properties by concatenating nouns with their corresponding prepositions, e.g. of, with, for, in, etc. For example, in Figure 2 a periphrastic relation is `fred:componentOf`. In fact, it is the result of the concatenation of the noun *"component"* and the preposition *"of"* as they appear in the input text of the CQ. Periphrastic relations are relevant in our scenario as they express frames as binary relations and not in terms of *n*-relations. They can be easily identified in the graph as they are OWL object properties defined by FRED using a local namespace. Equation 3 formalises the set of frames evoked by periphrastic relations as $F_{periphrastic}$, thus providing us a method to address the activity 3 in Figure 1. The local namespace[4] is used by FRED for producing URIs from the input textual elements. In Figure 2 the local namespace is associated with the predix `fred:`.

[4]The local namespace can be customised by a user in FRED either by means of its API or Web interface. We remark that FRED uses a number of namespaces and prefixes associated with external ontologies and vocabularies used for representing a text as RDF, e.g. DOLCE, VerbNet, DBpedia, etc. We refer the interested readers to [4] for more details about FRED.

$$F_{periphrastic} \equiv \{f |\ f \in \texttt{owl:ObjectProperty} \wedge namespace(R) = \texttt{fred:}\} \quad (3)$$

Hence, we explain how FrODO copes with activities 4 and 5 depicted in Figure 1 for periphrastic relation. In the case of a periphrastic relation the frame is the binary relation itself, which produces an object property, while the arguments are the subject and object of the relation that play the agentive and passive roles, respectively. The object property is generated with a naming convention that follows the template `:{relation}{ObjectClassName}` with a camel case notation. In this tamplate (i) `{relation}` is substituted by the name of the periphrastic relation produced by FRED, e.g. `componentOf`, and (ii) `ObjectClassName` is substituted by the name of the class typing the individual that plays the passive role in the relation, e.g. `System`. The object property `:componentOfSystem` is produced for the periphrastic relation in our running example. The naming convention is based on the assumption that arguments that play a passive role convey domain-peculiar knowledge. Then, two classes are generated from the subject and object of the periphrastic relation. The naming convention is based on the template `:{ClassName}`. In such a template `{ClassName}` is a variable to substitute with the actual value being the name of the class typing either the subject or the object of the periphrastic relation. For example, the classes `:Component` and `:System` are produced for the case represented by our example. Accordingly, the object properties resulting from periphrastic relations are enriched with domain and range axioms by following the same rationale adopted for the *n*-ary relation case. That is, for a given object property the domain and range are set to `owl:Thing` and the class produced from the individual being the object of a periphrastic relation, respectively. In our example, the range of the object property `:componentOfSystem` is `:System`. Furthermore, an inverse relation is materialised for each object property produced. The naming convention used for the inverse object properties is based on the template `:is{ObjectProperty}of` with a camel case notation. In such a template `{ObjectProperty}` is a variable which is substituted with the name of the object property that the current one is an inverse of. For example, is `:isComponentOfSystemOf` is the inverse produced for the object property `:componentOfSystem`. Then, the class generated from subject of the periphrastic relation is axiomatised with an existential restriction by applying the pattern defined in Equation 4.

$$S' \sqsubseteq \exists r_p.O', \qquad p \in F_{periphrastic}, s \in S, o \in O, p(s,o) \in G_{FRED} \quad (4)$$

In Equation 4 (i) p is a predicate that recognised as a periphrastic relation and holding between a subject s and an object o is a graph G_{FRED}, e.g. the triple $\langle$`fred:component_1`, `fred:componentOf`, `fred:system_1`$\rangle$; (ii) S is the class used as the type for the subject s, e.g. `fred:Component`; (iii) O is the class used as the type for the object o, e.g. `fred:System`; S' and O' are the classes generated by FrODO for S and O, e.g. `:Component` and `:System`; (iv) r_p is the binary property produced by FrODO from the input periphrastic relation, e.g. `:isComponentOfSystemOf`.

Finally, FrODO adds `rdfs:label` annotations to generated classes and properties. The following is the OWL produced by FrODO from the periphrastic relation occurring in our running example.

```
ObjectProperty: componentOfSystem
    Annotations: rdfs:label "component of system"
    Domain: owl:Thing
    Range: System
    InverseOf: isComponentOfSystemOf
ObjectProperty: isComponentOfSystemOf
    Annotations: rdfs:label "is component of system of"
    Domain: System
    Range: owl:Thing
    InverseOf: componentOfSystem
Class: Component
    Annotations: rdfs:label "Component"
    SubClassOf: componentOfSystem some System
Class: System
    Annotations:  rdfs:label "System"@en
```

FrODO is implemented as a Python Web application. Its source code is available on a GitHub repository[5] and a running instance is available online[6].

4. Evaluation

4.1. Experimental setup

We designed our experiment as a user-based study in order to address *RQ1* and *RQ2*. Namely, we asked a number of participants to perform an ontology design task starting from a set of identified CQs used as ontological requirements. The ontology design task was composed of two conditions. The first condition, i.e. *C1*, aimed at designing an ontology able to answer the provided CQs without the support of FrODO. On the contrary, the second condition, i.e. *C2*, aimed at designing an ontology able to answer the provided CQs with the support of FrODO. In both conditions the target ontology editor was Protégé. In case of the condition C2 the participants were asked to copy a CQ from the list of CQs they were provided with and paste it into the Web interface of FrODO. Then, they were asked to (i) execute FrODO in order to get the resulting ontology and (ii) import such an ontology into Protégé. This operation had to be executed once per each CQ the participants were provided with. No strategy on the order of CQs to solve for both condition C1 and C2 was imposed to the participants, i.e. they could start from any of the provided CQs by following their preferred order. Additionally, in case of condition C2 (with FrODO), the participants could opt for the implementation strategy the felt more confident with. This means, for instance, a participant might first import all the ontology modules returned by FrODO for each CQs and then refine those modules in the final ontology as a whole. Similarly, a participant might import and refine the ontology modules produced by FrODO for each CQs one-by-one incrementally. For the selection of the CQs we first identified three common CQ patterns out of the set of 12 CQ patterns observed in literature by [22]. The selected CQ patterns capture actors, relations, quantities, and modalities, as well as temporal and spatial elements. Namely the three CQ patterns are:

[5] https://github.com/anuzzolese/frodo
[6] https://w3id.org/stlab/frodo that redirects to http://semantics.istc.cnr.it/frodo

Table 1. CQs used as sample in our experiment.

Set	ID	CQ	Pattern
S1	CQ1	When is the level of a chemical substance recorded in a water body?	P1
	CQ2	What is a parameter that represents the quality of water bodies?	P2
	CQ3	Who records the amount of microbiological substances in surface waters in time?	P3
S2	CQ4	What are the contaminated sites in a geographical area recorded in time?	P1
	CQ5	When is the rate of hospitalisation related to a disease registered?	P2
	CQ6	Who monitors the hospitalisations for a disease in geographical area?	P3

- *"When is [object] [relation]?"*, i.e. pattern P1;
- *"What is [object] [relation] [object]?"*, i.e. pattern P2;
- *"Who [relation] [object]?"*, i.e. pattern P3.

For each of the three patterns we randomly picked two CQs from the corpus of CQs we defined in the context of the WHOW project[7]. This allowed us to get two sets of CQs, i.e. *S1* and *S2* having three CQs each. Table 1 reports the CQs selected along with their corresponding set, identifier, and pattern.

We recruited 7 participants for the experiment with all of them being experts in ontology design and patterns. Each participant was asked to perform C1, i.e. ontology design without FrODO, with one of the two CQ sets and C2, i.e. ontology design with FrODO, with the other CQ set. The assignment of the CQ sets to the participants with respect to the two ontology design conditions was performed in order to have the same CQs in S1 and S2 modelled alternatively with FrODO or without FrODO by a a balanced number of participants. Hence, the participants were divided into two groups, i.e. (i) *Group 1* tackling the CQs in the set S1 under the experiment condition C2 and the CQs in the set S2 under the experiment condition C1 and (ii) *Group 2* tackling the CQs in the set S1 under the experiment condition C1 and the CQs in the set S2 under the experiment condition C2. Additionally, we asked 3 participants to carry out the condition C2 as first option and then the condition C1 as second option. The remaining 4 participants were asked to carry out C1 first and then C2. The rationale of this choice was to mitigate possible biases introduced in the experiment by the order that the two conditions were executed by the participants. In fact, a participant, while addressing the experiment condition requested to be the first in the sequence, might acquire, for example, a deeper knowledge of the domain or she might benefit from the re-use of one or more ontology design patterns identified and used in the condition tackled as first option. This might make the condition tackled as second option easier to be solved, thus affecting the veracity of the results. The participants were supervised by an evaluator, who was in charge of (i) introducing FrODO with a brief demonstration, (ii) providing a detailed explanation of the experiment to the participants, (iii) supporting the participants during the experiments, (iv) recording the times taken by the participants for solving conditions C1 and C2. At the end of the experiment each participant was supposed to produce two ontologies as output. That is, one ontology addressing the CQs associated with condition

[7]The Water Health Open knoWledge (WHOW) is a project co-financed by the Connecting European Facilities of the European Union under grant agreement 2019-EU-IA-0089. Project website: https://whowproject.eu/

Table 2. Structural metrics used for assessing the ontologies produced by the participants to the experiment for both condition C1 and C2.

Metric	Description
# of annotation assertions	The total number of annotations in the ontology. Values are on ordinal scale.
# of axioms	The total number of axioms defined for classes, properties, datatype definitions, assertions and annotations. Values are on ordinal scale.
# of classes	The total number of classes defined in the ontology network. Values are on ordinal scale.
# of datatype properties	The total number of datatype properties defined in the ontology network. Values are on ordinal scale.
# of inverse object properties	The total number of object properties having a `owl:inverseOf` axiom for representing their inverse properties. Values are on ordinal scale.
# of logical axioms	The axioms which affect the logical meaning of the ontology. Values are on ordinal scale.
# of object properties	The total number of object properties defined in the ontology. Values are on ordinal scale.
# of object property domain axioms	The total number of axioms specifying the domain of an object property. Values are on ordinal scale.
# of object property range axioms	The total number of axioms specifying the range of an object property. Values are on ordinal scale.
# of SubClassOf axioms	The total number of `rdfs:subClassOf` axioms defined in the ontology.
Axiom/class ratio	The ratio between axioms and classes computed as the average amount of axioms per class. Values are computed as $\frac{\#\text{ of axioms}}{\#\text{ of classes}}$.
Class/property ratio	The ratio between the number of classes and the number of properties. Values are computed as $\frac{\#\text{ of classes}}{\#\text{ of properties}}$.
Inverse relations ratio	The ratio between the number of inverse relations and all the relations defined in the ontology. Values are on a scale ranging fron 0 to 1 and are computed as $\frac{\#\text{ of inv. object properties}+\#\text{ of inv. funct. datatype properties}}{\#\text{ of object properties}+\#\text{ of datatype properties}}$.
Inheritance Richness	The average number of subclasses per class computed as proposed by [23]. Inheritance Richness is expressed on ordinal scale.
Relationship Richness	The ratio between non-inheritance relations and the total number of relations defined in the ontology as proposed by [23]. Inheritance relations are `rdfs:subClassOf` axioms. Values are on a scale ranging from 0 (i.e. the ontology contains inheritance relationships only) to 1 (i.e. the ontology contains non-inheritance relationships only).

C1 and another ontology addressing the CQs associated with condition C2. We remark that the CQs associated with C1 or C2 were either those available for set S1 or set S2.

At the end of the experiment we asked the participants to rate ten statements using a five-point Likert scale (from 1: Strongly Disagree to 5: Strongly Agree). The ten statements are those of the System Usability Scale (SUS) [24]. The SUS is a well-known metric used for evaluating the usability of a system. It has the advantage of being technology-independent, and reliable even with a very small sample size [25] as in our case. It also

provides a two-factors orthogonal structure, which can be used to score the scale on independent Usability and Learnability dimensions [25]. The adoption of SUS in our experiment was not meant for assessing neither the usability of FrODO per se nor its integration with Protégé. On the contrary, it was meant for investigating the effectiveness of FrODO and its implemented methodology in supporting ontologists in ontology design tasks.

The ontologies produced by participants were evaluated with respect to the logical and structural dimensions as identified in the ontology evaluation framework formalised by [26]. The logical dimension was assessed by detecting the lack of inconsistencies by means of a DL reasoner. The DL reasoner we opted for is HermiT[8]. The structural dimension was assessed with different metrics that have been defined and used in literature [26,23,27,28]. The structural metrics we used are reported in Table 2. We do not assess the functional dimension, which is the ability of an ontology to address requirements and cover the domain [26]. This is because all the ontologies result from expert ontology engineers that were asked to model ontologies able to address the proposed CQs that identify the target ontological requirements. Hence, we assume that all resulting ontologies are qualitative from the functional perspective.

4.2. Results

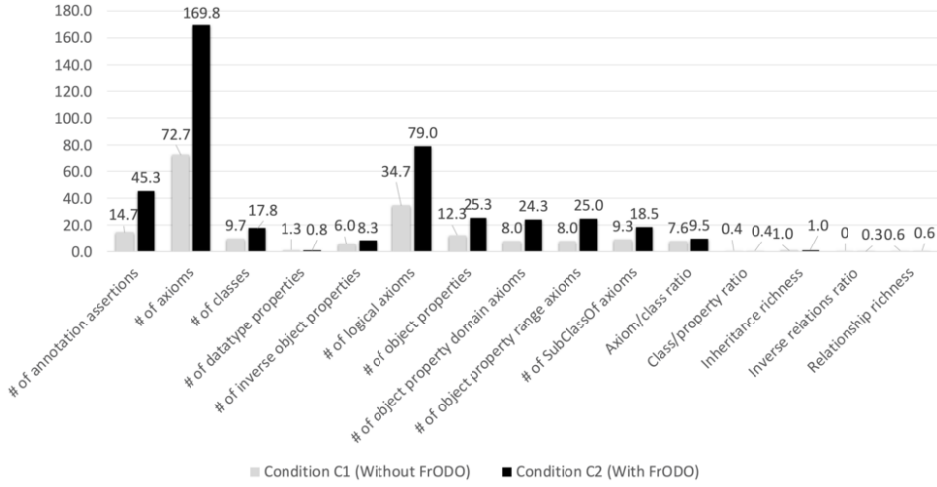
All ontologies designed by participants for conditions C1 and C2 are logically consistent. Instead, Figures 3a and 3b show the results computed for the metrics reported in Table 2 for participant groups 1 and 2, respectively. The values reported are averaged among those obtained for each participant to the experiment with regards to the experiment condition. We used OntoMetrics[9] [27] as a tool for computing such metrics automatically. Most metrics we took into account perform better in the experiment condition that includes FrODO (i.e. condition C2) than in the other does not (i.e. condition C2). This observation is valid for both participant groups 1 and 2.

Figure 4a reports the times taken on average by participants to complete the tasks associated with the conditions C1 and C2 regardless to the execution order. Standard deviation values are reported among brackets for each series. On the contrary, Figure 4b reports the same times by taking into account the specific order in which the participants tackled C1 first and then C2. In the first case (cf. Figure 4a) the times recorded are comparable, on average, for the participants of group 1, i.e. 39 minutes for C1 Vs 38.3 for C2. Nevertheless, for the participants of group 2 we recorded longer times when the ontology design process was supported by FrODO, i.e. 32 minutes recorded for condition C2 Vs. 19.7 minutes recorded for condition C1. However, if we limit our analysis to the case having the condition C2 executed after the condition C1 (cf. Figure 4b), then we observe longer times when the ontology design process was not supported by FrODO. That is, for group 1, we recorded on average 37 minutes taken for C2 Vs. 56.5 taken for C1 and, for group 2, we recorded on average 21 minutes taken for C2 Vs. 24 taken for C1.

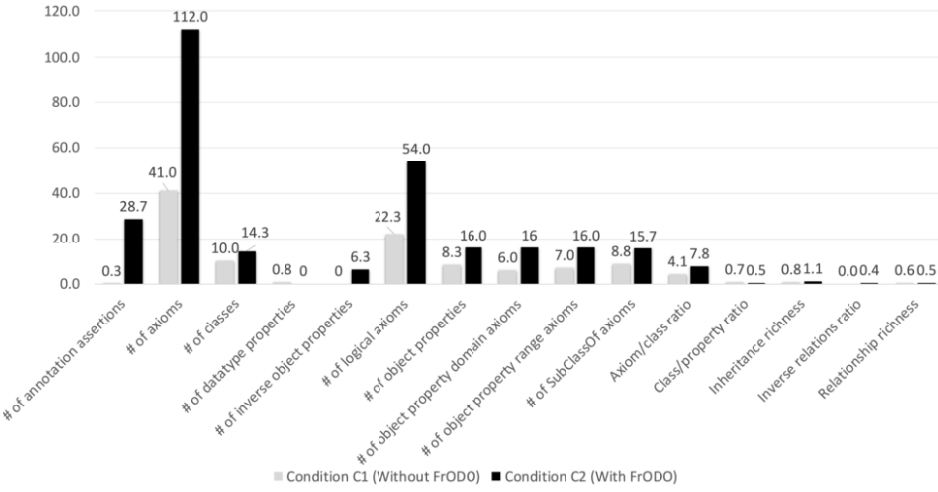
Figure 5 reports the SUS scores in terms of the overall SUS score and its two orthogonal indicators, i.e. Learnability and Usability. Standard deviation values are reported

[8] http://www.hermit-reasoner.com/: last visited on May 2022.

[9] https://ontometrics.informatik.uni-rostock.de/ontologymetrics/ last visited on May 2022.



(a) Group 1

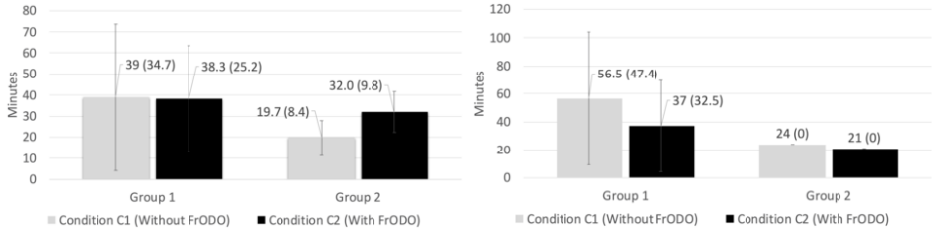


(b) Group 2

Figure 3. Results computed for all structural metrics reported in Table 2

among brackets. We report separates perspectives over the results obtained with the SUS along with the global scores by distinguishing between two cases: (a) the condition C2 (with FrODO) was executed before C1 (without FrODO) and (b) vice versa, the condition C1 (without FrODO) was executed before C2 (with FrODO). For the global perspectives the sores are 73.9, 85.87, and 69.9 for SUS, Learnability, and Usability, respectively. For perspective (a) the sores are 68, 82, and 63.3 for SUS, Learnability, and Usability, respectively. Finally, for perspective (b) the sores are 88.8, 95, and 86.6 for SUS, Learnability, and Usability, respectively. All the ontologies generated by the participants for both conditions C1 and C2 are available on GitHub[10]. Simirlarly, the CSV data con-

[10] https://github.com/anuzzolese/frodo/tree/main/evaluation/ontologies



(a) All participants.

(b) Participants that tackled Condition 2 after Condition 1.

Figure 4. Times required on average for completing the tasks associated with condition 1 and 2, respectively. The times are expressed in minutes.

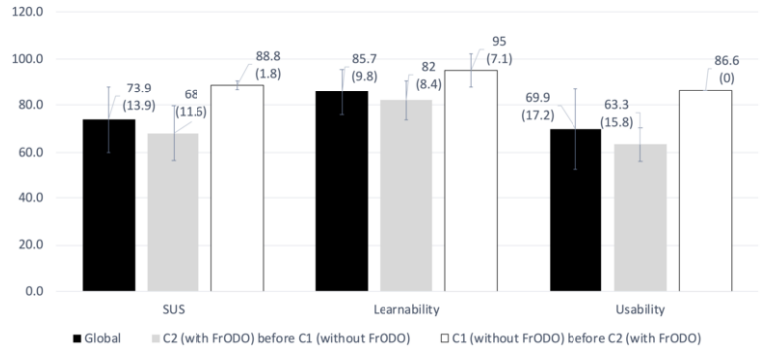


Figure 5. SUS, learnability and usability scores.

taining the metrics computed with OntoMetrics, times, and SUS results are published on Zenodo [11].

4.3. Discussion

The analysis based on the structural metrics computed over the ontologies generated for cases C1 and C2 suggests that the ontologies produced with the support of FrODO (i.e. case C2) are richer in terms of (i) axioms, (ii) annotations, (iii) classes, (iv) taxonomic relations (i.e. `rdfs:subClassOf` relations), (v) object properties, and (vi) inverse object properties. This richness is confirmed by analysing the results for the class/property ratio, inheritance richness, inverse relation ratio, and relationship richness. In fact, for all these metrics we recorded higher values from the experiment condition involving FrODO (i.e. condition C2) into the development process for both participant groups 1 and 2. This addresses *RQ1*. The SUS-based analysis shows that an ontology design methodology based on FrODO can be considered usable regardless to the specific order in which conditions C1 and C2 were set in the experiment. In fact, based on empirical studies [46], a SUS score of 68 represents the average usability value even in case of a small number of participants [25]. Additionally, we observed, on average, that the adoption of FrODO in

[11] https://doi.org/10.5281/zenodo.6574273

an ontology design workflow is not time-consuming. This is satisfactory and addresses *RQ2*.

5. Conclusions and future work

In this work we present the Frame-based Ontology Design Outlet (FrODO), which is a method and tool able to draft ontologies from competency questions (CQs) automatically. First, we provide a background about ontology design methodologies and automatic solutions for generating ontologies from text. Then, we describe the method implemented by FrODO that builds on top of FRED for leveraging frame semantics to gather domain knowledge and formalise it into OWL ontology drafts. Those drafts can be used by ontology engineers to make agile ontology design methodologies (e.g. XD, SAMOD, etc.) smoother. The effectiveness of FrODO was assessed by means of a user-based study that involved 7 participants all of them being expert ontology engineers. The aim of the a study was twofold, that is, evaluating: (i) the quality of the ontology drafts produced by FrODO from the logical and structural perspectives; and (ii) the usability of FrODO when used for an ontology design task. The experiment shows that FrODO produces richer ontologies if compared to the ontologies designed without the support of FrODO from the same set of CQs. The System Usability Scale (SUS) shows excellent usability scores for all the perspective we analysed. Future works include the evaluation of the ontologies produced with the support of FrODO from the functional perspective. Then, we aim at designing and developing a plug-in for Protégé able to embed FrODO inside the popular ontology design framework, thus strengthening the cohesion between the two systems. Additionally, we plan to extend FrODO in order to provide better support to the generation of datatype properties, inverse functional datatype properties, and disjoint axioms, which are overlooked in the current version of the tool. Among the others the implementation of disjoint axioms will benefit from the alignment with foundational ontologies, such as DOLCE+DnS Ultralite, which is already provided by FRED.

Acknowledgements

This work has been supported by the Water Health Open knoWledge (WHOW) project co-financed by the Connecting European Facility programme of the European Union under grant agreement INEA/CEF/ICT/A2019/206322.

References

[1] Grüninger M, Fox MS. The role of competency questions in enterprise engineering. In: Benchmarking—Theory and practice. Springer; 1995. p. 22-31.

[2] Presutti V, Daga E, Gangemi A, Blomqvist E. eXtreme Design with Content Ontology Design Patterns. In: Blomqvist E, Sandkuhl K, Scharffe F, Svátek V, editors. Proc. of WOP 2009. vol. 516 of CEUR Workshop Proceedings. CEUR-WS.org; 2009. .

[3] Peroni S. A Simplified Agile Methodology for Ontology Development. In: Dragoni M, Poveda-Villalón M, Jiménez-Ruiz E, editors. Proc of OWLED 2016. vol. 10161 of Lecture Notes in Computer Science. Springer; 2016. p. 55-69. DOI: 10.6084/M9.FIGSHARE.3189769.V2.

[4] Gangemi A, Presutti V, Recupero DR, Nuzzolese AG, Draicchio F, Mongiovì M. Semantic Web Machine Reading with FRED. Semantic Web. 2017;8(6):873-93. DOI: 10.3233/SW-160240.
[5] Etzioni O, Banko M, Cafarella MJ. Machine Reading. In: AAAI Spring Symposium: Machine Reading. AAAI; 2007. p. 1-55. DOI: 10.5555/1597348.1597430.
[6] Fillmore CJ, et al. Frame semantics. Cognitive linguistics: Basic readings. 2006;34:373-400.
[7] Gangemi A, Presutti V. Ontology design patterns. In: Handbook on ontologies. Springer; 2009. p. 221-43.
[8] Grüninger M, Fox M. Methodology for the Design and Evaluation of Ontologies. In: IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing; 1995. .
[9] Sure Y, Staab S, Studer R. In: Staab S, Studer R, editors. On-To-Knowledge Methodology (OTKM). Berlin, Heidelberg: Springer Berlin Heidelberg; 2004. p. 117-32. DOI: 10.1007/978-3-540-24750-0_6.
[10] Keet CM, Lawrynowicz A. Test-Driven Development of Ontologies. In: Sack H, Blomqvist E, d'Aquin M, Ghidini C, Ponzetto SP, Lange C, editors. ESWC. vol. 9678 of Lecture Notes in Computer Science. Springer; 2016. p. 642-57. DOI: 10.1007/978-3-319-34129-3_39.
[11] Wiśniewski D, Potoniec J, Ławrynowicz A, Keet C. Analysis of Ontology Competency Questions and their formalisations in SPARQL-OWL. Journal of Web Semantics. 2019 11;59. DOI: 10.1016/j.websem.2019.100534.
[12] Carriero VA, Mariani F, Nuzzolese AG, Pasqual V, Presutti V. Agile Knowledge Graph Testing with TESTaLOD. In: ISWC Satellites; 2019. p. 221-4.
[13] Wisniewski D. Automatic translation of competency questions into sparql-owl queries. In: Companion Proceedings of the The Web Conference 2018; 2018. p. 855-9. DOI: 10.1145/3184558.3186575.
[14] Cimiano P. Ontology learning and population from text: algorithms, evaluation and applications. vol. 27. Springer Science & Business Media; 2006.
[15] Al-Aswadi FN, Chan HY, Gan KH. Automatic ontology construction from text: a review from shallow to deep learning trend. Artificial Intelligence Review. 2020;53(6):3901-28. DOI: 10.1007/s10462-019-09782-9.
[16] Cimiano P, Völker J. A framework for ontology learning and data-driven change discovery. In: Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB). Springer; 2005. p. 227-38. DOI: 10.1007/11428817_21.
[17] Witte R, Khamis N, Rilling J. Flexible ontology population from text: The owlexporter. In: Proceedings of LREC 2010; 2010. .
[18] Tanev H, Magnini B. Weakly supervised approaches for ontology population. In: Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics; 2006. p. 17-24.
[19] Hearst M. Automatic acquisition of hyponyms from large text corpora in proc. In: 14th International Conference Computational Linguistics, Nantes France; 1992. .
[20] Völker J, Rudolph S. Lexico-Logical Acquisition of OWL DL Axioms - An Integrated Approach to Ontology Refinement. In: Proceedings of ICFCA 2008. vol. 4933 of Lecture Notes in Artificial Intelligence; 2008. .
[21] Falco R, Gangemi A, Peroni S, Shotton DM, Vitali F. Modelling OWL Ontologies with Graffoo. In: Presutti V, Blomqvist E, Troncy R, Sack H, Papadakis I, Tordai A, editors. ESWC (Satellite Events). vol. 8798 of Lecture Notes in Computer Science. Springer; 2014. p. 320-5. 10.1007/978-3-319-11955-7_42.
[22] Ren Y, Parvizi A, Mellish C, Pan JZ, Deemter Kv, Stevens R. Towards competency question-driven ontology authoring. In: European Semantic Web Conference. Springer; 2014. p. 752-67.
[23] Tartir S, Arpinar IB, Sheth AP. Ontological evaluation and validation. In: Theory and applications of ontology: Computer applications. Springer; 2010. p. 115-30. DOI: 10.1007/978-90-481-8847-5_5.
[24] Brooke J. SUS - A quick and dirty usability scale. Usability evaluation in industry. 1996;189(194):4-7. DOI: 10.1201/9781498710411-35.
[25] Sauro J. A practical guide to the system usability scale: Background, benchmarks & best practices. Measuring Usability LCC; 2011.
[26] Gangemi A, Catenacci C, Ciaramita M, Lehmann J. Modelling ontology evaluation and validation. In: European Semantic Web Conference. Springer; 2006. p. 140-54. DOI: 10.1007/11762256_13.
[27] Lantow B. OntoMetrics: Putting Metrics into Use for Ontology Evaluation. In: KEOD; 2016. p. 186-91. DOI: 10.5220/0006084601860191.
[28] Carriero VA, Gangemi A, Mancinelli ML, Nuzzolese AG, Presutti V, Veninata C. Pattern-based design applied to cultural heritage knowledge graphs. Semantic Web. 2021;12(2):313-57. DOI: 10.3233/SW-200422.