Simplifying Ontology Design Pattern Implementation

Giacomo De Colle[1,3], Rachel A. Mavrovich[1,3], Cristian Keroles[1], Elena Milivinti[1,3], Giorgio A. Ubbiali[2], Fabricio Rodrigues[1], Noor M. Al-Khaouli[1] and John Beverley[1,3]

[1]University at Buffalo, 135 Park Hall, Buffalo, NY 14260-4150, USA
[2]PTH Brixen College, Seminarplatz 4 39042 Brixen, Italy
[3]National Center for Ontological Research, 500 Seneca Street, Buffalo NY 14208, USA

Abstract

Ontology design patterns can be defined as reusable solutions for implementing and developing ontologies. In this paper, we focus on ontology design patterns as tools to help structure data in an ontology-conformant way, and we present an architecture that facilitates the standardized creation of content design patterns of high quality. To achieve this, we first present a collection of desiderata as minimal quality requirements for effective patterns. To facilitate the adoption of these quality standards, we then present a standardized workflow and resources to easily generate design patterns and the related data, queries and documentation. These resources include Python scripts to translate graphical representation of patterns into instance-level data, as well as templates to write effective documentation and to prompt LLMs to aid in the creation of SPARQL queries. We demonstrate the results of this work by applying our approach to three different use cases and three different ontologies — the Common Core Ontologies, the PHASES ontology, and the D3FEND ontology. First results show promising portability of the approach across at least three different projects, and the resulting pipeline aims to be generalizable across domains and different ontological approaches.

Keywords

Ontology Design Patterns, CCO, D3FEND, PHASES, LLMs

1. Introduction

Design patterns (DPs) have long served as a cornerstone of knowledge engineering, offering reusable, modular solutions to recurring modeling problems [1]. The term was originally adopted in software engineering, where patterns were understood as named, documented solutions to problems that arise repeatedly in a given design context [2]. Transferred to the domain of ontology development, this idea proved especially productive: rather than building ontologies from scratch each time, engineers could draw on a repertoire of established modeling solutions, adapting them to local needs while preserving their core structure. Ontology Design Patterns (ODPs) emerged from this insight as structured templates that guide the construction of semantically precise, interoperable knowledge representations [3, 4]. By encoding particular modeling choices into reusable artifacts, ODPs reduce redundancy, promote consistency across projects, and lower the barrier of entry for ontology engineers working in unfamiliar domains. Their adoption has been advocated across a range of knowledge engineering contexts, from the development of domain ontologies to the construction of knowledge graphs.

Despite their recognized value, ODPs remain substantially under-documented and weakly validated in practice [5]. Lack of explicit use cases, formal competency questions or systematic evaluation criteria make it difficult to assess whether a given ODP is fit for purpose in a new context. Patterns are frequently shared as isolated files, leaving prospective reusers with little guidance on the assumptions embedded in the pattern, the modeling problems it is designed to address, or the conditions under which it can be safely extended or specialized [6]. This problem is especially acute where semantic precision and cross-system interoperability are at stake. Recurring modeling errors or suboptimal

Proceedings of FOIS 2026 Satellite events co-located with the 16th International Conference on Formal Ontology in Information Systems (FOIS 2026), September 21-25, 2026, Vitória, Brazil.

gdecolle@buffalo.edu (G. D. Colle); rmavrovi@buffalo.edu (R. A. Mavrovich); ckeroles@buffalo.edu (C. Keroles); emilivin@buffalo.edu (E. Milivinti); giorgio.ubbiali@pthsta.it (G. A. Ubbiali); fr33@buffalo.edu (F. Rodrigues); nooralkh@buffalo.edu (N. M. Al-Khaouli); johnbeve@buffalo.edu (J. Beverley)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-27

CEUR-WS.org/Vol-4342/paper9.pdf

choices that superficially resemble valid solutions further complicate the landscape by showing that poor documentation does not merely limit reuse but can actively mislead it [5].

We identify two broad scenarios in which ODPs emerge. In the first, patterns are deliberately constructed from identified modeling issues, through the systematization of repeated practice across distinct settings, the extrapolation of modeling expertise to recognize a new problem as an instance of a recurring type, or the analysis of structural regularities within a target domain. In the second scenario, patterns are revealed retrospectively in existing ontologies developed without explicit use cases or competency questions, through approaches such as inspecting documentation produced during the ontology development, domain-driven analysis, or reverse engineering of the ontology's repeatable structures.

This distinction carries direct consequences for how patterns should be documented and validated: a pattern constructed around a known modeling problem can be evaluated against that problem directly and explicitly, whereas a pattern extracted from an existing ontology must first be assessed for whether it corresponds to a genuine, recurrent modeling need in the first place. For a discussion of a similar issue in the creation of competency questions, see [7]. In both scenarios, however, the core challenge is the same — moving from an implicit or informal modeling solution to a fully specified, testable, and reusable artifact.

This challenge is amplified by the scale at which ontology engineering and implementation operate. As ontologies are increasingly expected to interoperate across institutional and disciplinary boundaries, the demand for well-documented, rigorously validated ODPs is greater. Moreover, as long as ontologies are used to encode and structure datasets, for example to produce knowledge graphs, they need to be seamlessly shared across software architectures for their implementation. Manual approaches to pattern documentation and testing, while valuable, are time-consuming and difficult to scale. There is therefore a pressing need for methods and tools that can support — though not replace — manual human intervention to identify, specify, and validate ODPs systematically.

Tools such as standardized documentation and templates, scripting and Large Language Models (LLMs) offer a timely opportunity to address this deficit. Recent advances in LLM capabilities have demonstrated their effectiveness in a range of knowledge engineering tasks, including ontology verbalization, entity extraction, and the generation of natural language descriptions of formal structures [8]. Applied to ODP development, LLMs can assist in easing the creation and testing of ODPs, structured templates can help standardize and facilitate the creation of documentation and rigorous visual implementation, and scripts can simplify and automate testing. Rather than replacing human judgment, these tools function here as a semi-automated layer within a broader pattern engineering workflow, one that makes rigorous ODP development more tractable at scale.

This paper proposes and evaluates a semi-automated workflow for ODP generation, implementation and documentation. Section 2 reviews the existing literature on ODPs, tracing the current spectrum of available positions, and proposes a working definition suited to this work. Section 3 presents the method through which we developed the proposed workflow. Section 4 describes in detail the workflow and the tools produced to standardize it, implement it and facilitate its adoption. Moreover, we report on three case studies in which we applied the workflow, drawing on three different ontologies: the Common Core Ontology (CCO), a widely used mid-level suite of ontologies [9], D3FEND, an ontology for cybersecurity and defensive techniques [10], and the ontology for Promoting Health Aging through Semantic Enrichment of Solitude Research (PHASES) [11]. Section 5 discusses the results, limitations, and directions for future work. We conclude in Section 6.

The contribution provided by this work is multi-faceted: on the one hand, we present a general discussion on the role of ODPs in knowledge graph engineering and we propose minimal quality standards that are needed to operationalize ODPs in this context. On the other, we showcase an open-source set of tools, templates, examples and standards that can be used to easily build a high-quality ODP alongside with accompanying queries and visual and written documentation. The tools introduced in this paper lower the barrier of entry for the community of ontologists and knowledge graph engineers interested in creating ontology-compliant graphs and operationalizing them in a practical scenario. Finally, we provide worked out examples of how this workflow was used and implemented among first

test-cases coming from different existing projects.

2. Literature Review

Scholarship on ODPs addresses what ODPs are, specifies the requirements a pattern should satisfy, and develops methods for testing, validating, and disseminating ODPs for reuse [12]. Read together, the contributions form a spectrum of positions ranging from foundational definitions to practical engineering concerns. Tracing that spectrum clarifies both what the field has achieved and where the gaps still remain.

In the Semantic Web literature, ODPs are treated as compact, self-contained ontology fragments intended as reusable solutions to recurring modeling problems. Gangemi's [3] account establishes this baseline: patterns are building blocks for Semantic Web content, each linking a recurring problem type to a modeling solution, and their value lies precisely in being reusable across contexts without requiring users to start from scratch.

A classification of ODPs is proposed within the NeOn project [13], whose main typologies are: 1) content ODPs, focused on modelling and domain issues, 2) reasoning ODPs, focused on ontology reasoning and standardization, 3) correspondence ODPs, focused on ontology engineering and alignment, 4) structural ODPs, focused on ontology structure and language, 5) lexicoSyntactic ODPs, focused on harmonizing language and ontology meanings, and 6) presentation ODPs, focused on ontology annotation and graphical representation [4, 6]. In these accounts, ODPs should be reusable without forcing users to inherit all commitments of the ontology they originate from, and they should be communicated as reusable artifacts rather than as isolated files [14].

Research works on the field of automatic ontology construction ascribe to ODPs sufficient precision and regularity to support machine-assisted reuse, not only human interpretation. This is made explicit in pattern-based methodology discussions that emphasize patterns as repeatable modeling solutions and as components in a broader engineering workflow [4]. ODPs are embedded in a structured workflow: requirements are collected, candidate patterns are identified and evaluated against those requirements, and selected patterns are assembled and extended to produce the ontology.

A challenge within this workflow concerns the compatibility of patterns drawn from different sources. Works on pattern alignment treat ODPs as reusable components that may nonetheless be heterogeneous in structure, assumptions, and modeling idioms, and therefore require systematic mappings to be jointly usable [15]. Building on this, work on specialization strategies shows that reuse is rarely a simple copy-paste operation: a pattern often needs controlled variation—e.g., how properties are specialized, constrained, or propagated—so that it fits local modeling goals without breaking desirable inferences [6]. Taken together, these contributions deepen the "reusable building block" picture: patterns are not mere "Lego blocks"; they have important compatibility constraints [16, 6].

ODP literature also largely emphasizes systematic pattern validation [16, 4]. Explicit requirements, such as competency questions that the ontology must answer, make it possible to determine in advance a candidate pattern addresses a given use case, and make the fit between pattern and problem something that can be assessed and documented rather than merely asserted. This points in the direction of a broader applicability of ODPs: improving interoperability, facilitating reuse and fostering cleaner data integration. Sales and Guizzardi [5] raise concerns about modeling failures, i.e., recurring errors or suboptimal choices that can undermine ontology quality.

In summary, the literature treats ODPs as: (i) small modular solution units [3, 14], (ii) a systematic design methodology that stabilizes modeling choices [16, 4], (iii) components in a broader ecosystem requiring alignment and controlled specialization [16, 6], and (iv) targets of evaluation in both positive [17] and negative (anti-pattern) forms [5].

For the purposes of this paper, we define an ODP as a small, modular, reusable fragment of an ontology that has example instances and that offers a standardized template to create instances as a solution to a recurring modeling problem, specified against explicit use cases and intended reuse conditions. From here on, unless specified otherwise, in this paper we will use "ODP" to refer to this working

definition. In the current landscape of literature and open-source projects there seems to be a lack of tools that can help practitioners to more easily create ODPs. Our aim is to establish a workflow-ready account of ODPs that can be implemented in a semi-automated pipeline for ontology implementation and documentation.

3. Methods

At the beginning of this work, we convened to discuss the difficulties arising with documenting and using large ontologies such as CCO. Despite the presence of whitepapers and documentations, many ontologies leave partially open how they should be practically used to construct a conformant and standardized knowledge graph, and the presentation and documentation of ODPs are the natural solutions to this issue. We then proceeded to the literature review described in Section 2. What we identified as lacking was the presence of open-source resources that could be easily adopted by practitioners to facilitate the creation of ODPs, specifically in the context of RDF [18] and OWL ontologies [19]. We first produced a generalized workflow and corresponding high-level, minimal quality-criteria that we found necessary for the practical implementation of ODPs in existing ontology projects. This high-level workflow is described in Section 3.1. We then proceeded to the creation of a set of tools to facilitate the implementation of this workflow and to its testing. We justify our adoption of software and languages in Section 3.2 and we describe the implemented workflow and testing in Section 4.

3.1. General Methodology

The aim of this project is to facilitate the creation of ODPs that are able to address the following concerns:

1. Assist in the creation of ontology-conformant knowledge graphs.
2. Act as visual and written documentation for users implementing the ontology.
3. Assist in the creation of queries that are able to retrieve information from the graphs.

The first item in this list is particularly important for this work: we are chiefly interested in using ODPs as exemplary knowledge graphs, reusable templates to instantiate ontology classes. In order to achieve the three results in the list and to allow for the ODP to be testable, assessable, and deployable within a broader ontology/knowledge graph engineering pipeline, we identified five quality standards that we consider to be minimally required for the creation, testing and documentation of an ODP:

1. A collection of competency questions that the ODP needs to address.
2. A computational representation of the competency questions as queries in a formal query language.
3. An RDF serialization of sample instances used to test the queries.
4. A graphical representation of the ODP for the communication and validation purposes.
5. The resources created to satisfy these points are all created following a standardized formatting and style that allows them to be interchangeable and interoperable with other humans and machines.

Regarding **quality standard 1**, as described in [20, 21], competency questions can be one of the starting and leading elements needed to guide ontology development. Similarly, ODPs should be led by competency questions so that it is driven by stakeholder needs — specifically, by the questions that users want to ask of their data. This is especially true if ODPs are taken as guidelines to insert instance-level data in an ontology-conformant way into a knowledge graph. If the resulting knowledge graph needs to be able to answer certain competency questions, then it is through the ODP structuring the knowledge graph that they are able to do that or not. In the rest of this paper, we refer to these



Figure 1: Representation of the intended generalized workflow.

works on competency questions for discussions on the quality control and modality of collection of competency questions and we do not expand on it any further [20, 21].

Regarding **quality standard 2**, often competency questions are collected from stakeholders and domain experts in natural language. To be operationalized, they need to be encoded in a computable query language that can run against the ontology-conformant dataset. Translating competency questions into a query language strongly facilitates the process through which an ODP can be reliably tested for its capability to answer such questions.

Regarding **quality standard 3**, a simple way to ensure that the queries are working as intended is to run them against sample data that is at least partially representative of the type of data that the final knowledge graph will be storing. This can be achieved, for example, if the ODP is structured as an example graph, providing not just classes and relations between classes but also examples of how they are instantiated and related. In this way, it provides a baseline to generate a synthetic graph to test the queries. Reasoners such as HermiT [22] can also be run over such a graph to identify unintended consequences of the ODP. All these steps require not just the creation of the ODP as a graphical rendition or an OWL fragment, but rather the presence of a proper RDF graph that instantiates examples of data following the ODP.

Regarding **quality standard 4**, a graphical representation is of fundamental importance for communicating the ODP with stakeholders, domain experts and other ontologists. It is furthermore important that this representation is unambiguous, clearly comprehensible to all the interested parties, and faithful to the eventual RDF encoding that will need to be implemented in the knowledge graph. This is necessary, although not sufficient, to ensure that the intended interpretation of the ODP is retained across iterations involving other professionals, and that this interpretation is faithfully implemented in the RDF data.

Regarding **quality standard 5**, the presence of a standardized syntax that allows for the ODPs to be consistently produced, shared and exported into different languages is crucial in order to create a replicable workflow, computationally manipulate the ODPs and create a pipeline that handles them in a predictable and deterministic way. Quality standard 5 is especially needed to make sure that the ODPs are interoperable and reusable within other parts of software and data architectures and by other ontologists. These proposed minimal quality standards form a framework in which each element constrains and validates the others. These requirements are not necessarily meant to cover all the aspects regarding the quality of ODPs, but rather focus on those aspects that are important for their implementation as guides to structure knowledge graphs.

As part of this project, we developed a documented workflow that users can adopt in order to produce an ODP that satisfies these requirements. Section 4 describes in detail the tools and resources that we developed to implement it, facilitate its adoption and partially automate its deployment, as well as three test cases to illustrate its implementation.

1. The ontologist collects competency questions (**step 1** in Figure 1). As already discussed, competency questions should act as the starting input for the process. These competency questions are ideally produced by domain experts, end users, or other practitioners working alongside the ontologist. If this is not possible, the competency questions can also be generated by the ontologist following a plausible use case or to showcase a particular intended feature of the ontology. The competency questions should be collected in such a way that they can realistically be asked of a dataset after being encoded in a query language. This step is completed once the

competency questions collected represent a suitable guide for the use case.

2. The ontologist proceeds to the creation of the ODP to answer the competency questions using a graphical representation tool (**step 2** in Figure 1). In this step, the conceptual structure of the ODP is effectively created. The terms used in creating the ODP are selected from the ontology that the ontologist considers the most appropriate to represent the domain. At this stage, the OWL axioms associated with the relations and classes used in the ODP are often useful guides. Take axioms introduced using rdfs:subClassOf, owl:equivalentClass or rdfs:subPropertyOf. Arguably, in ontology practices these are used to signal edges that instance nodes of the corresponding class should have at all times, and are thus chief guides for the creation of ODPs that act as templates for the creation of instance-level graphs. For example, saying that `ex:car` is `rdfs:subclassOf` `ex:has_part` some `ex:wheel` usually means that we can expect a correct ontology-compliant graph to have an instance of `ex:car` connected with `ex:has_part` at least one `ex:wheel`. Strictly speaking, OWL does not require the graph to have these instances as for example, SHACL does. Moreover, this does not necessarily mean that an ODP needs to include all the relations identified through the OWL axioms of a certain class, as this would often result in an overly verbose ODP, but rather that they can be used as starting guidelines to identify elements of the intended ODP if the ontologist is working with a well-axiomatized ontology.
3. The ontologist proceeds to the creation of a testing dataset (**step 3** in Figure 1). Recall that the main use of ODPs in this project is to create ontology-conformant knowledge graphs that are able to answer competency questions. Being able to create a knowledge graph that corresponds to the ODP is thus a fundamental requirement. This testing dataset can be created using synthetic or real-world data. While real-world data offers a more realistic test bed for the ODP, obtaining it and translating it into the knowledge graph can often become a costly procedure which is best executed at a later stage. If the ontologist is testing the structure of the ODP rather than its specific content, using synthetic data can also be perfectly acceptable. This step is completed once the graph corresponding to the ODP is created. The ontologist can now ensure the consistency of the ODP using tools such as reasoners.
4. The ontologist proceeds to the creation of queries corresponding to the competency questions collected in step 1 (**step 4** in Figure 1). This can be implemented by a query language of choice of the ontologist, which needs to be selected on the basis of the environment that the knowledge graph is supposed to be deployed in. The step is completed once the queries are able to retrieve the data that the answers the competency questions.
5. Once the queries are created, the ontologist can use them to test whether the ODP is able to answer the competency questions (**step 5** in Figure 1). This is done by running the queries against the dataset corresponding to the ODP and created in step 3. This step is completed once the queries are reliably displaying the data as requested by the competency questions that were used as input at the beginning of the workflow. If the queries are not reliably retrieving such data, the preceding steps are to be repeated as needed.

3.2. Implementation and Testing Methodology

In Section 4 we describe how we implemented the workflow we just presented and how we developed tools that facilitate its adoption. We now briefly justify our choice of programming languages and standards. The visual representations of diagrams are encoded using the Mermaid diagramming language [23], an open-source language that generates diagrams from text. Mermaid was chosen insofar as it can be manipulated both through text-based and graphical means, which makes it flexible enough to cater to different users. Moreover, it can be easily manipulated and generated by AI models such as LLMs. We also created standardized Mermaid templates that can be reused to easily generate such diagrams. The queries to answer competency questions are operationalized as SPARQL queries, a W3C recommendation and a pattern-matching query language that can be used to manipulate and extract information in RDF graphs [24]. Since for our tests we decided to use ontologies that are saved in OWL and can be used to create RDF graphs, we considered SPARQL as the more appropriate query language

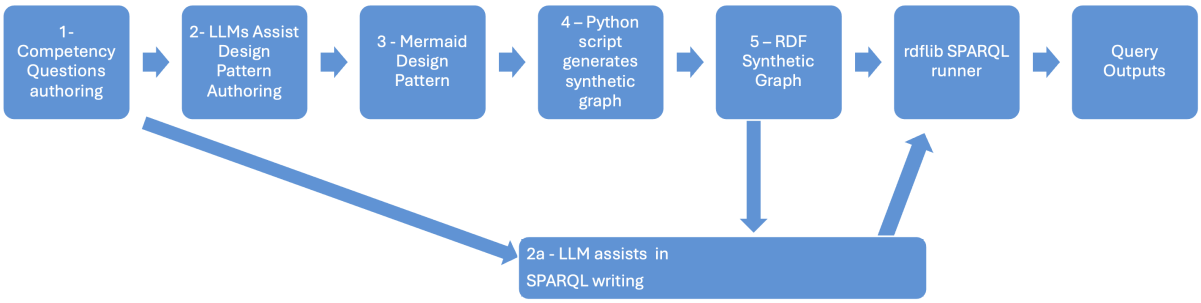

Figure 2: Representation of the implemented workflow.

to adopt. SPARQL is a standardized language, able to produce clearly understandable queries and even to perform targeted reasoning, and was thus a natural choice given that we want our queries to be able to take full advantage of OWL. The scripts described in the results section were written using Python and the RDFlib library [25]. The example graphs were saved as files in the Turtle format [26], which we chose because of its simplicity and human readability. We decided to use an RDFLib Python script to run the queries insofar as we considered it a lightweight mode to rapidly test them. The aim of this work is to produce and test ODPs in a simple way, rather than implementing them in a large-scale application or software environment for which a graph database such as Apache Jena might be more appropriate [27]. We furthermore created Markdown standardized templates that can be reused to simplify the creation of documentation for ODPs. These templates are then ingested by the Python scripts to simplify the creation of ODPs and to create prompts for LLMs. In our workflow we also adopted LLMs as described in the next section, and specifically ChatGPT 5.1 and Claude Sonnet 4.6, as we wanted to test whether these widely adopted models could be usefully implemented in assisting with the creation of ODPs. We reserve to do in the future more research into whether different languages or different LLM architectures provide better results.

The workflow described in Section 4 was tested against three ODPs created for three different ontologies. CCO is a set of eleven modular ontologies widely used in the US government sector for applications in intelligence and defense. PHASES is an ontology developed to connect studies in the domains of gerotranscendence and solitude. Both PHASES and CCO are developed using the Basic Formal Ontology (BFO) [28, 29] as top-level ontology. The third ontology we use is D3FEND, developed by MITRE corporation to encode domain knowledge on cybersecurity infrastructure and cyberdefensive techniques. D3FEND doesn't adopt any top-level ontology, although it is mapped to BFO [30]. We decided to adopt three different ontologies in order to diversify the type of tests to run, so that our work wouldn't be biased towards one specific type of modeling proper of one domain. These ontologies are moreover all in the process of being built or being further expanded, so that their community of users can benefit from the discussion and possibility of standardization coming from ODPs development.

Development of the resources, including the Python scripts, markdown templates and Mermaid ODPs was conducted on GitHub and is available under an open-source license at [31].

4. Results

We now describe how we created a semi-automated and standardized workflow that implements the more general workflow described in Section 3; in Section 4.1 we describe how we created a series of tools and template to facilitate its adoption and in Section 4.2 we describe how three different test cases show promising generalizability of our methodology.

4.1. Design Pattern Implementation

1. The ontologist collects competency questions (**step 1** in Figure 2). We provide a standardized markdown textual template for the ontologist to write this content, which includes 1) competency questions, 2) a description of the ontological resources used to create the ODP, and 3) a description of the content of the ODP. This template acts not only as a way to store the competency questions, but also as part of the documentation of the ODP.
2. The ontologist proceeds to the creation of the ODP to answer the competency questions using the Mermaid diagramming language to represent it graphically (**steps 2** and **3** in Figure 2). We require for this diagram to follow certain standardized rules and style guidelines which are attached in the accompanying material. Most notably, we require that the Mermaid file references the URIs used by the ontology classes and relations and that it unambiguously distinguishes between classes, relations and instances. The ontologist can choose to proceed to create the Mermaid ODP manually, or by leveraging an LLM. Given that Mermaid is a text-based language, the Mermaid template can easily be copied and pasted to generate the graph manually, or can be used by the LLM as guidance to produce syntactically correct Mermaid outputs with a prompt like the following:

   > Please generate a Mermaid graph that describes {insert description of the pattern} using the attached Mermaid template as guidance to distinguish between instances, classes and relations {insert Mermaid template }.

   The resulting Mermaid file can be automatically rendered in a graphical format by using the built-in preview function of text editors such as VSCode and GitHub, or by using plugins or dedicated websites such as mermaid.live [32].
3. The ontologist proceeds to the creation of an RDF synthetic dataset corresponding to the ODP (**steps 4** and **5** in Figure 2). In order to facilitate this procedure, we created a Python script that automatically translates the Mermaid graph into RDF. This is possible in a reliable way only insofar as the ontologist follows the guidelines provided in step 1, so that the syntax of the Mermaid graph is standardized enough for the script to properly recognize ontology URIs and relations between the elements in the diagram.
4. The ontologist proceeds to the creation of SPARQL queries corresponding to the competency questions collected in step 1 (**step 2A** in Figure 2). This step can be largely facilitated by the usage of LLMs, if correctly prompted. In order to do this, we provide a Python script that takes as input the RDF graph generated in the previous step and the competency questions, then proceeds to create as output a text which contains both and accompanies it with the correct instructions that can be passed to the LLM to prompt it to create the corresponding SPARQL output. The template used is the following:

   > Please write SPARQL queries to answer these competency questions:
   > {competency questions}
   > Assume you run the queries against this graph
   > {ttl graph}
   > Make sure that the SPARQL queries you create are generalizable, and are not using hard-coded individuals from the example graph.

   where the content of "competency questions" and "ttl graph" are automatically filled by the provided Python script, using the resources created in the previous step as starting material.
5. Once the SPARQL queries are created, the ontologist can use them to test whether the ODP is able to answer the competency questions (steps 6 and 7 in Figure 1). This is done by running the SPARQL queries against the synthetic example graph corresponding to the ODP. In order to facilitate this, we provide a standardized Python script that runs the SPARQL queries against the synthetic graph and prints the results. Once the ontologist is satisfied with the SPARQL queries produced in the previous step, they need only to copy-paste them to the script to run them and test whether the ODP is able to answer the needed questions.

Each step can be repeated and iterated over until a satisfactory result is reached. For steps 2 and 3 in Figure 1, a first viable result is the creation of a syntactically correct Mermaid graph, which can be automatically graphically rendered and which is following the requirements of the template discussed in point 2. This pattern can later be refined for semantic clarity and coherence. For steps 4 and 5, the ontologist should iterate until they are able to create a syntactically correct RDF graph with the example instances and the correct classes and relations contained in the Mermaid graph. If the Mermaid is correctly formatted according to the template and instructions provided, then this step can be automatically performed thanks to the Python script provided. If this step is not successful, this is likely due to errors in the formatting of the Mermaid graph produced in steps 2 and 3, and the ontologist needs to reiterate previous steps until they are able to produce a correct RDF graph. Step 2A is successful insofar as it allows the ontologist to perform steps 6 and 7, i.e., to run the SPARQL queries against the RDF graph. If the queries are not able to be run or to produce the expected results, this can be due to errors in the syntax of the Mermaid or RDF, or in some modelling pitfall in steps 2 and 3, and the ontologist needs to reiterate on them until the SPARQL produces the intended results. Even when the SPARQL queries are able to answer the competency questions, the ontologist needs to manually inspect whether the query generated by the LLM is generalizable outside the current examples, or whether it is for example hard-coding the names of instances in the queries in order to get the intended results. If this final step is also passed, then the ODP can be considered validated.

Table 1
Patterns developed and validated for testing purposes.

Step	Input	Output	Type	Validation Requirement
Competency question authoring	Markdown template	Competency questions and documentation	Manual	Completed document
Design pattern authoring	Competency questions documentation, OWL axioms, Ontology file	Mermaid Diagram	LLM-assisted	The diagram syntax is correct; the Mermaid template is followed
Synthetic graph generation	Mermaid diagram	Synthetic .ttl graph	Script-automated	The .ttl file is syntactically valid; optional reasoner validation
SPARQL queries prompt	Synthetic .ttl graph, Competency questions documentation	SPARQL queries prompt	Script-automated	The prompt is generated and can be used as input for the LLM
SPARQL queries generation	SPARQL queries prompt	SPARQL queries	LLM-assisted	The queries are syntactically correct and correspond to the competency questions
SPARQL queries testing	SPARQL queries, Synthetic .ttl graph	Queries results	Script-automated	The queries are retrieving the data as expected

4.2. Test Cases

To evaluate the proposed methodology and contribute to ongoing development and standardization processes of three different ontology projects, we decided to test our workflow to produce three ODPS from three different ontologies. These ontologies include the previously introduced CCO, D3FEND and PHASES. For each pattern, the full workflow was executed, going from competency questions to Mermaid, translated to RDF, SPARQL query generation, and finally query-based validation. The authors of this paper created the ODPs following the described methodology. Nevertheless, the content of the ODPs was discussed in detail with members external to the author group, as described in detail in each section. The PHASES pattern is currently part of the PHASES documentation [33] and the CCO pattern is under review to be included in the official CCO documentation [34].

Table 2
Patterns developed and validated for testing purposes.

Pattern name	Ontology	No. CQs & SPARQL queries	Validation outcome
Facility Location	CCO	3	Valid
Password Change	D3FEND	4	Valid
Narrative Assessment	PHASES	3	Valid

The patterns were selected to represent a range of domains, so that our tests could be differentiated enough and avoided reproducing biases or styles inherent to only one type of ontology. Table 2 provides a summary of the patterns tested, with the number of competency questions, SPARQL queries, and validation outcome for each. The Password Change and Narrative Assessment ODPs are quite large and cannot be easily represented in graphical format in this paper. We only included a graphical representation for the Facility Location ODP, faithfully recreated to better fit the formatting of this paper.

For each ODP, competency questions were collected. Each pattern is accompanied by additional context documents which show the intended purpose of the ODP, its role within the broader ontology architecture and related modules. Each pattern was then modelled using Mermaid, representing the intended instance diagram and relations between said classes. Mermaid was then used to produce a graphical representation of the pattern. The Mermaid file was then used to automatically generate an RDF synthetic graph. SPARQL queries were generated using LLMs to assist in the creation and were then manually refined. One example of these queries is reproduced in the paper for the Password Change ODP, although it would have been too verbose to include all of them. The ODPs were then validated by running the SPARQL queries against the synthetic graph. As such, each of these ODPs meets the quality standards described above and can be used both as documentation for the respective ontology and as a tool to build and query a knowledge graph following the ontology.

4.2.1. Facility Location

This ODP describes location relationships in which material entities are situated in geographic regions or sites and how these sites can be transitively located one in another. For example, a certain building located in New York City is also located in New York State. The ODP was developed as part of a broader standardization effort within the CCO development team, and was conducted with input from one CCO developer. The pattern is currently under review for inclusion as part of the official CCO documentation.

The competency questions defined for this pattern are reverse-engineered from existing CCO documentation and from the content of the ODP:

1. Where is a specific facility located? For example, Facility 1 is directly located in City 1 (“NY City”).
2. What larger site contains this facility, and are there smaller sites that the facility is also located in that fall within that larger site? For example, Facility 1 is located in City 1 (“NY City”), which is located in State 1 (“NY State”).
3. What are all the facilities located within a particular site? For example, for City 1, Facility 1.

The pattern was modelled using the Basic Formal Ontology (BFO) relation `located_in` (`BFO_0000171`) and the class `site` (`BFO_0000029`) and CCO class `facility` (`ont00000192`). Figure 3 depicts the pattern visually.

4.2.2. Password Change

This ODP describes a plausible cybersecurity scenario. An account is created and its password is changed. The D3FEND ontology is used to represent the agent creating the account, the agent creating the password and the password change. Then, using the axioms in the ontology, we were able to identify possible attacks that could be inferred from the presence of this password change event as well as possible

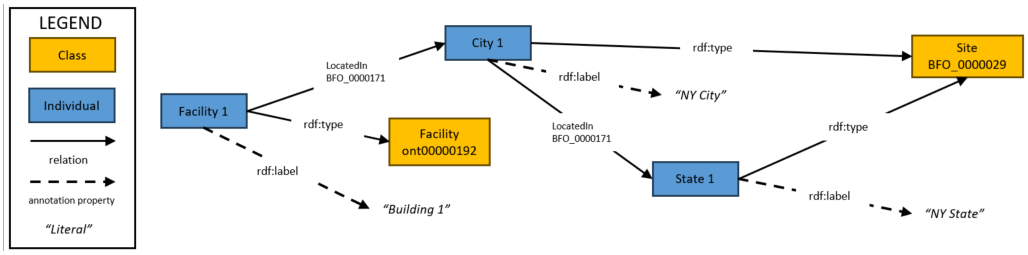

Figure 3: Facility Location ODP, rendered using the standardized Mermaid template as guide and clearly distinguishing between instances, classes and literals.

cyberdefensive techniques that could be implemented in order to protect the password from other attacks. The ODP was discussed with one member of the D3FEND developers team and their comments were adopted in this version, although this is not officially part of the D3FEND documentation.

The core event of password change is represented using `d3f:UserAccountPasswordChangeEvent` and is connected to an instance of Account Manipulation (`d3f:T1098`) to represent a possible cyberattack that can compromise it. The password is an instance of `d3f:Credential` and cyberdefensive techniques such as `d3f:One-timePassword` are related through it via the relation `d3f:strengthens`. The competency questions for this ODP were reverse-engineered on the basis of the ODP and the axioms of the ontology:

1. What agent created this account? This can be queried by identifying the `d3f:UserAccountCreationEvent` which is related through `d3f:hasParticipant` to a `d3f:hasAgent`.
2. What defensive actions have or can be taken to protect this account and its password? This can be queried by identifying entities of the types `d3f:UserAccount` and `d3f:Credential` and then the defensive actions connected to them through `d3f:strengthens` and `d3f:restricts`.
3. What agent changed the account password? This can be queried by identifying the `d3f:UserAccountPasswordChangeEvent` which is related through `d3f:hasParticipant` to a `d3f:hasAgent`.
4. What offensive actions may be associated with the password change event? This can be queried by identifying the `d3f:UserAccountPasswordChangeEvent` which is connected through `d3f:maybeassociatedwith` to instances of `d3f:OffensiveTechnique` or its subclasses.

For illustrative purposes, the LLM-generated and user-reviewed SPARQL query corresponding to the first competency question is the following:

```
SELECT DISTINCT ?agent ?agentLabel ?account ?accountLabel
WHERE { ?event rdf:type d3f:UserAccountCreationEvent ;
d3f:hasAgent ?agent ;
d3f:hasParticipant ?account .
?account rdf:type d3f:UserAccount .
OPTIONAL { ?agent rdfs:label ?agentLabel }
OPTIONAL { ?account rdfs:label ?accountLabel }}
```

4.2.3. Narrative Assessment — PHASES

This ODP describes the process through which the subject of an investigation in the field of narrative identity is interrogated by a narrative identity analyst to assess what the recurring themes and structure of their life's narrative are. The competency questions and content of the ODP were discussed with one narrative identity domain expert, who was iteratively involved in the assessment and evaluation process. The ODP was then presented and discussed during meetings of the PHASES project and revised according to suggestions for acceptance in the project. Currently, this is part of the official PHASES documentation. Resources from PHASES, as well as other OBO Foundry ontologies [35], were used.

The ODP represents an individual participating in a process in which they generate a narration of their lives. The narrative identity expert then assesses the themes and structure present in this narrative. The competency questions identified after discussion with the domain expert are:

1. What agent is creating a narrative? This can be queried by identifying the Subject (MF:0000016) who is holding Participant Under Investigation Role (OBI:0000097) and which is connected through Realizes (BFO:0000055) to the Narrative Construction Process (BFO:0000015).
2. What theme(s) are identifiable in the narrative? This can be queried by identifying the Narrative Theme Assessment, an instance of Conclusion Based on Data (OBI:0001909), that is connected to the Narrative through the relation isAbout (IAO:0000030).
3. What is the identifiable structure of the narrative of the agent? This can be queried by identifying the Narrative Theme Assessment, an instance of Conclusion Based on Data (OBI:0001909), that is connected to the Narrative through the relation isAbout (IAO:0000030).

5. Discussion

The literature described in Section 2 has paved the way to the discussion of ODPs in creating ontologies. In this work, we rather focus on how to use ODPs to help assist ontologists in the creation of standardized knowledge graphs. The main contribution of this work is the creation of an open toolkit that can be used to simplify the creation of understandable, implementable and easily shareable ODPs. The quality standards discussed in section 3 address whether the ODP can be reliably communicated to other members of the team and stakeholders and whether the ODP can be used to answer their competency questions; other considerations on the quality of the ODP, on how to use ODPs as building blocks for ontologies, and other related topics are outside the scope of this work.

Two of the examples discussed in Section 4 make use of ontologies developed using BFO. While the third is not directly developed using a top-level ontology, it is mapped to BFO. Nevertheless, the methodology discussed here can be easily implemented using other ontologies, including other top-level ontologies. In order to implement our work, we chose to adopt RDF and SPARQL as languages to store and query data. Nevertheless, there is in principle no reason why other architectures cannot also be used to implement this workflow, even though the Python script presented in Section 4 is written to only work with RDF.

The work presented here is the first step in the implementation of a broader project, and possesses certain inherent limitations. LLMs in this project are used as tools that integrate or connect already well-understood and deterministic processes, as described in [36]. In the workflow implementation discussed in Section 4, LLMs are used to facilitate the creation of patterns, reducing manual labor in repetitive tasks such as creating a SPARQL or Mermaid file that are syntactically correct. The creation of ODPs using LLMs is, in our experience, a good way to create a first representation of the domain and to facilitate the creation of a syntactically correct Mermaid graph. Nevertheless, it is still unreliable if meant to be used to create a semantically sound graph. Even when provided with the ontology file as part of their prompt, the LLMs we used (ChatGPT 5.1 and Claude Sonnet 4.6) routinely misused the terms in the ontology. In comparison, we were able to achieve much better results in the creation of SPARQL queries to operationalize competency questions. Using both the competency questions and the sample graph as input, as discussed in Section 4, the LLM is suitably constrained to the ODP context and is able to create useful outputs. A trend we noticed was that the LLM was often "taking shortcuts", for example by creating queries that hard-coded identifiers for the individuals we created in the synthetic graphs. These queries are not generalizable outside of the synthetic graph and can not be reused in the future. To alleviate this problem, the prompt template we used and present in Section 4 is also indicating to the LLM not to use hard-coded individuals.

A natural follow-up for this project is to test whether it is possible to use the Mermaid diagrams presented in this paper to map structured and unstructured data to the ODP, thus further automating the generation of ontology-compliant knowledge graphs. One example of a similar project can be seen in [37], which uses ODPs as elements for an ETL pipeline. One other important expansion of this project

is to further develop and automate the whole pipeline, for example by introducing an API call to an LLM service or by simplifying other steps besides the ones which are already automated. Finally, more research is needed in enhancing the reliability of the LLMs in producing and executing the various steps.

6. Conclusion

In this paper we have presented tools to implement a workflow to facilitate the generation of ODPs used to generate knowledge graphs. These tools include templates to produce standardized Mermaid diagrams, automatically translate them to RDF synthetic datasets, assist in the LLM-driven generation of SPARQL queries and run the queries. We tested the workflow against three different use cases and found that it substantially facilitated the process and allowed for the creation of clearly documented ODPs. Future work will include a better integration of LLMs in the process and the expansion to mapping of non-RDF data into design patterns.

Declaration on Generative AI

The authors declare that they have used Generative AI, and specifically ChatGPT 5.1 and Claude Sonnet 4.6, to assist in the writing of the code included in the supplementary material as well as the stylistic polish and formatting of certain sections of the paper. Grammarly and the Free AI Paraphrasing Tool were also used in order to check grammar and spelling, paraphrase, and reword. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content. Generative AI was also used as part of the methodology as discussed in the paper.

Funding

The PHASES portions of this paper were supported by the National Institute of Aging for the 1U01AG088074-01 Promoting Health Aging through Semantic Enrichment of Solitude Research (PHASES) project.

The research of Giorgio Ubbiali on this paper has been supported by a grant from the Autonomous Province of Bolzano–South Tyrol within the project "Climate, Plastics and Sustainability: Ontology and Operationalisation" (ClimOO) under the auspices of the Klimaplan 2040, grant number 19/2025.

Acknowledgments

The authors wish to thank Finn Wilson and Samuel Eskew for their engagement with early versions of this project; the members of the CCO developers group and in particular Alan Ruttenberg for their feedback on the facility location design pattern; the members of the PHASES project and in particular Bill Duncan for their feedback on the PHASES pattern; the members of the D3FEND-CCO group for their feedback on the password design pattern.

References

[1] Hitzler P, Gangemi A, Janowicz K, Krisnadhi AA, Presutti V. Ontology Engineering with Ontology Design Patterns: Foundations and Applications. NLD: IOS Press; 2016.

[2] Gamma E, Helm R, Johnson R, Vlissides J, Booch G. Design Patterns: Elements of Reusable Object-Oriented Software. Boston, Mass.: Addison-Wesley Professional; 2011.

[3] Gangemi A. Ontology Design Patterns for Semantic Web Content. In: Gil Y, Motta E, Benjamins VR, Musen MA, editors. The Semantic Web — ISWC 2005. Berlin, Heidelberg: Springer; 2005. p. 262–76. doi:10.1007/11574620_21

[4] Presutti V, Blomqvist E, Daga E, Gangemi A. Pattern-Based Ontology Design. In: Suárez-Figueroa MC et al., editors. Ontology Engineering in a Networked World. Berlin, Heidelberg: Springer; 2012. p. 35–64. doi:10.1007/978-3-642-24794-1_3
[5] Sales TP, Guizzardi G. "Is It a Fleet or a Collection of Ships?": Ontological Anti-patterns in the Modeling of Part-Whole Relations. In: Advances in Databases and Information Systems. Cham: Springer; 2017. p. 28–41.
[6] Hammar K. Ontology Design Pattern Property Specialisation Strategies. In: Knowledge Engineering and Knowledge Management. Cham: Springer; 2014. p. 165–80. doi:10.1007/978-3-319-13704-9_13
[7] Wisniewski D, Potoniec J, Lawrynowicz A, Keet CM. Competency Questions and SPARQL-OWL Queries Dataset and Analysis. ArXiv, abs/1811.09529. 2018.
[8] Negro A, Kus V, Futia G, Montagna F. Knowledge graphs and LLMs in action. Shelter Island: Manning Publications; 2025.
[9] Jensen M, De Colle G, Kindya S, More C, Cox AP, Beverley J. The Common Core Ontologies. Front Artif Intell Appl. 2024 Apr 27;394:47–58.
[10] Kaloroumakis PE, Smith MJ. Toward a knowledge graph of cybersecurity countermeasures. 2021.
[11] Beverley J, Bowker JC, Reischer HN, Mavrovich RA, Hurley R, Kindya S, et al. Towards Healthy Aging through Semantic Enrichment. In: Proceedings of JOWO, co-located with FOIS 2025; 2025 Sep 8–9; Catania, Italy.
[12] Hammar K, Lin F, Tarasov V. Information Reuse and Interoperability with Ontology Patterns and Linked Data. In: Business Information Systems Workshops. Berlin, Heidelberg: Springer; 2010. p. 168–79. doi:10.1007/978-3-642-15402-7_23
[13] NeOn Project. Ontology Design Patterns [Internet]. [cited 2026 Mar 23]. Available from: http://ontologydesignpatterns.org/index.php/Ontology_Design_Patterns_._org_(ODP)
[14] Gangemi A, Presutti V. Ontology Design Patterns. In: Staab S, Studer R, editors. Handbook on Ontologies. Berlin, Heidelberg: Springer; 2009. p. 221–43. doi:10.1007/978-3-540-92673-3_10
[15] Rodriguez-Castro B, Ge M, Hepp M. Alignment of Ontology Design Patterns. In: On the Move to Meaningful Internet Systems: OTM 2012. Berlin, Heidelberg: Springer; 2012. p. 682–99. doi:10.1007/978-3-642-33615-7_16
[16] Blomqvist E, Sandkuhl K. Patterns in Ontology Engineering: Classification of Ontology Patterns. In: Proceedings of the 7th International Conference on Enterprise Information Systems. Miami: SciTePress; 2005. p. 413–6. doi:10.5220/0002518804130416
[17] Mortensen JM, Horridge M, Musen MA, Noy NF. Applications of Ontology Design Patterns in Biomedical Ontologies. AMIA Annu Symp Proc. 2012;2012:643–52. PMID: 23304337.
[18] World Wide Web Consortium (W3C). Resource Description Framework (RDF). Available at: https://www.w3.org/RDF/
[19] World Wide Web Consortium (W3C). Web Ontology Language (OWL). Available at: https://www.w3.org/OWL/
[20] Keet CM, Khan ZC. On the Roles of Competency Questions in Ontology Engineering. In: Knowledge Engineering and Knowledge Management. Cham: Springer Nature Switzerland; 2025. p. 123–32. doi:10.1007/978-3-031-77792-9_8
[21] Keet CM, Khan ZC. Characterising Competency Questions for Ontologies.
[22] Shearer R, Motik B, Horrocks I. HermiT: a highly-efficient OWL reasoner. In: Proceedings of OWLED 2008 EU; 2008 Oct 26–27; Karlsruhe, Germany. CEUR Workshop Proceedings. Vol. 432.
[23] Mermaid contributors. Mermaid [Internet]. GitHub; [cited 2026 Mar 23]. Available from: https://github.com/mermaid-js/mermaid
[24] World Wide Web Consortium (W3C). SPARQL 1.1. Available at: https://www.w3.org/TR/sparql11-query/
[25] RDFLib contributors. RDFLib [Internet]. GitHub; [cited 2026 Mar 23]. Available from: https://github.com/RDFLib/rdflib
[26] World Wide Web Consortium (W3C). RDF 1.1 Turtle. Available at: https://www.w3.org/TR/turtle/
[27] Apache Jena contributors. Apache Jena [Internet]. Available at: https://jena.apache.org/

[28] Arp R, Smith B, Spear A. Building Ontologies with Basic Formal Ontology. Cambridge, MA: MIT Press; 2015.
[29] Otte J, Beverley J, Ruttenberg A. BFO: Basic Formal Ontology. Applied Ontology. 2022;17(1):17–43.
[30] De Colle G., Kaloroumakis P., Introducing D3FEND-CCO. Forthcoming in Proceedings of the Semantic Technologies for Intelligence, Defense and Security, 2026
[31] Design Patterns Pipeline repository. Available from: https://github.com/rmavrovich/Designpatternspipeline
[32] mermaid.live contributors. mermaid.live [Internet]. Available from: https://mermaid.live
[33] PHASES repository. [Internet]. Available from: https://github.com/Buffalo-Ontology-Group/phases
[34] Common Core Ontology repository pull request. [Internet]. Available from: https://github.com/CommonCoreOntology/CommonCoreOntologies/pull/775
[35] Smith B, et al. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nat Biotechnol. 2007;25:1251–5.
[36] Giabbanelli PJ, Beverley J, David I, Tolk A. From over-reliance to smart integration: using large-language models as translators between specialized modeling and simulation tools. In: Proceedings of WSC 2025. 2025. p. 1119–30.
[37] Ontology Tradecraft: Project 4 [Internet]. GitHub; [cited 2026 Mar 23]. Available from: https://github.com/Applied-Ontology-Education/Ontology-Tradecraft/tree/main/projects/project-4