OntoForms: Automated Ontology-Driven Form Generation With Description Logic Reasoning

Bruno Szilagyi*, Edelweis Rohrer* and Regina Motz*

Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, 11300 Montevideo, Uruguay

Abstract

This paper presents a software component that generates a user interface structure for populating a domain ontology. The core of this work is an algorithm that takes an ontology and returns a structure describing the user interface. The component also provides functions for populating the ontology and editing existing individuals. Unlike previous approaches, this method can be implemented without any configuration. Additionally, it offers an easy-to-use configuration mechanism that allows irrelevant classes to be hidden and automatically populated. What distinguishes this work is that, instead of exploring the ontology using syntactic methods or queries, our algorithm employs services that implement description logic inference mechanisms. This work illustrates the proposed approach using the well-known People ontology.

Keywords

Ontology, Ontology population, Reasoner

1. Introduction

Ontologies are proven modelling artefacts for conceptualizing different domains, such as education and health, providing a structured framework to describe them in terms of concepts, instances, and roles (binary relations), as well as constraints about them [1]. Ontologies promote understanding of the domain, facilitate the formulation of queries on domain data, and have the capability of entailing implicit knowledge from the explicitly declared. Hence, nowadays there are increasingly ontology-based applications that take advantage of the benefits of ontologies, mainly regarding their ability to represent the domain at a conceptual level, hiding the complexity and implementation details of data structures.

From the point of view of the usability quality attribute, an ontology-based application must provide the end user with a friendly interface to query and enter data from the conceptual view provided by the ontology. However, as new user needs arise, besides modifying or extending the ontology, the user interface must also evolve. To avoid modifying the application front-end and the logic that populates the ontology every time the ontology changes, there exist approaches that generate data entry forms from ontologies, favoring the maintainability and reusability of ontology-based applications.

The primary goal of this tool is to bridge the gap between formal ontologies and practical data entry interfaces. Specifically, it aims to:

- Automate form generation by interpreting ontological classes and properties, eliminating the need for manual form design.
- Ensure semantic accuracy by strictly enforcing constraints such as datatype restrictions, cardinality rules, and property characteristics.
- Enhance usability by structuring forms in an intuitive manner, making them accessible even to non-experts.

Proceedings of FOIS 2025 Satellite events co-located with the 15th International Conference on Formal Ontology in Information Systems (FOIS 2025), September 10-12, 2025, Catania, Italy

*Corresponding author.

† These authors contributed equally: Bruno Szilagyi, Edelweis Rohrer, Regina Motz

bruno.szilagyi.ibarra@gmail.com (B. Szilagyi); erohrer@fing.edu.uy (E. Rohrer); rmotz@fing.edu.uy (R. Motz)

0009-0004-0522-3876 (B. Szilagyi); 0000-0002-8257-5291 (E. Rohrer); 0000-0002-1426-562X (R. Motz)

- Support customization through administrator configuration, allowing domain-specific adjustments without altering the core ontology.
- Facilitate interoperability by integrating with web-based frameworks.

To guide the development process, the following research questions were considered:

1. Constraint Translation (RQ1): How can ontological constraints (e.g., required properties, value ranges) be best converted into form validation rules?
2. Relationship Representation (RQ2): What user interface patterns (e.g., nested forms, dynamic selectors) most effectively capture complex ontological relationships?

To systematically develop and validate this tool, we adopt a methodology that combines Design Science Research (DSR), based on the guidelines by Hevner (2004) [2], with Agile practices, as outlined in the Agile Design Science Research Methodology (ADSRM) proposed by Conboy et al. (2015) [3]. The process begins by analyzing existing challenges in ontology-based data entry, such as the labor-intensive nature of manual form creation, potential mismatches between form fields and ontological constraints and limited flexibility in adapting forms to evolving ontologies. Using insights from the problem analysis, we design the tool in iterative cycles, focusing on:

- Ontology Processing: Leveraging libraries like OWLAPI or RDFLib to parse and interpret ontological structures.
- Form Generation Logic: Translating classes and properties into form fields.
- Validation Rules: Automatically deriving input constraints from ontological restrictions.
- User Interface Assembly: Structuring forms into logical sections and ensuring responsive design.

Each iteration incorporates feedback from domain experts and end-users to refine functionality. The tool's effectiveness is assessed through:

- Functional Testing: Verifying that generated forms correctly reflect ontological rules.
- Usability Studies: Gathering feedback on form intuitiveness and efficiency.

The main contribution of the present work is the implementation of a solution, OntoForms, that provides a set of integrated services for ontology management, as well as configuration and generation of forms to populate ontologies. The key component of the solution is the algorithm that receives a domain ontology, and optionally a configuration, and returns a structure describing the user interface.

OntoForms provides domain applications with different endpoints that solve functionalities such as obtaining the interface structure and populating the ontology once the user enters the data. An application for the administrator user is also provided, which makes use of OntoForms endpoints to ontology uploading, and forms configuration and previsualization.

This approach stands out from most existing ones in that it is entirely domain expert-users oriented. Unlike traditional ontology editors (e.g., Protégé) that require expertise in ontology engineering, OntoForms is designed for non-ontology specialist users who need to populate and manage ontology-driven data effortlessly.

OntoForms supports fully automated form generation as well as customizable configurations through optional settings, allowing it to adapt to different use cases without imposing setup burdens. Fields and prompts are dynamically organized into logical sections based on the ontology's structure, creating an intuitive layout that mirrors domain experts' mental models. Most notably, the system handles complex ontological constructs seamlessly, such as automatically managing classes for n-ary relations (where n > 2), completely shielding users from the underlying technical complexities. This transparent instance handling ensures domain experts can focus solely on providing accurate data rather than navigating ontological intricacies. The behaviour of OntoForms is shown by applying it to the well-known People ontology.

The development is openly licensed, the code is available at:
https://github.com/brunoszilagyiibarra/ontoforms-backend and
https://github.com/brunoszilagyiibarra/ontoforms-admin.

The remainder of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 describes the architecture of the solution, whereas Section 4 describes the core algorithm implemented in OntoForms. Section 5 illustrates the approach by applying it to People ontology and Section 6 presents some conclusions and future work.

2. Related work

This section reviews existing approaches for generating user interfaces that enable domain experts-users who need to populate ontologies with data but lack expertise in ontology engineering to interact with ontologies efficiently. Unlike ontology editors such as Protégé, which are designed for ontology engineers, the solutions discussed here focus on simplifying data entry (ABox manipulation) for end users in domain-specific applications. We exclude traditional ontology-authoring tools, as their complexity makes them unsuitable for non-specialists.

The work of Gonçalves et al. [4] was conceived to generate UI forms for the health domain, even though it can be applied to any domain. The proposed system has three main components: an XML configuration file used for the generation of the web form, a domain ontology, and a data model that describes the form. This approach requires a considerable configuration effort from an administrator user since, in addition to the domain ontology, it is necessary to provide the XML file and also a data model to configure the form. This also affects the maintainability of the solution, as the configuration file must be reviewed when the domain ontology evolves. The generated form allows the user to enter data (ontology instances) but does not enable the update of data already entered. To use the provided implementation it is necessary to upgrade it because it uses an old technology.

The thesis of Liu [5] is also an ontology-based approach that relies on a domain ontology and a called "technology level ontology" which maps the domain ontology to the underlying data schema where the ontology is stored. The user interface is generated from the set of attributes of the ontology that the domain expert selects from an intuitive interface that represents a view of the domain ontology. However, the fact that two different ontologies must be maintained can be tedious for the administrator user. This work focuses on drawing the form but not on populating the ontology with the instances entered by the user. This approach also uses an outdated technology.

Based on the work of Gonçalves et al., the approach of Vcelak et al. [6] consists of the generation of web responsive forms from domain ontologies, for the retrieval and edition of RDF data, enabling both the insert and the update of instances already entered, and validating data types and cardinality. Instead of maintaining a configuration structure, ontology annotations are used for this purpose. Hence, the administrator user must have some background in ontologies. However, the configuration of the form does not appear as a tedious task since it consists of just adding annotations, which makes the solution quite maintainable. Even though the authors give a clear description of the approach, the implementation of the solution is not available.

Another approach similar to the one of Gonçalves et al. is the proposed by Rutesic et al. [7], for the domain of pizzas. In addition to the domain ontology FOODON, an ontology describing the graphical user interface is a key component of the approach. On the one hand, it is a simple implementation that only requires changing the interface ontology when the domain ontology changes, and that generates a user-friendly interface. In addition to introducing new instances, this interface also allows new classes or properties to be added to the ontology. This can be critical when used by users with no knowledge of ontology design. On the other hand, the configuration is too manual for the administrator user, and the update of instances is not well resolved. The approach is implemented using up-to-date technology.

Within the manufacturing domain, the quality of manufacturing service capability (MSC) information can be measured by its accuracy and semantic interoperability. For this, the use of ontologies that describe MSC information contributes to a correct understanding among all stakeholders, avoiding ambiguity. To help manufacturing domain experts to provide MSC information using the ontology vocabulary, Peng at al. [8] propose the Ontology-based eXtensible Dynamic Form (OXDF) user interface architecture. Some relevant functionalities of this approach are the generation of forms from ontologies, a search

engine to find ontology entities, and a mechanism for the user to insert and update instances, as well as to define new classes or properties. This approach does not allow the configuration of the form by selecting the classes or properties that are presented or hidden. As the generated interface presents the whole ontology, the solution is easy to evolve. However, it requires domain experts to have some background on ontologies. The implementation of the approach is not available.

The work of Rocchetti et al. [9] presents a general ontology-based approach that is applied to the art domain. Besides configuring some aspects such as the set of key ontology concepts from which the form is generated, a graph representing the ontology structure is constructed as an intermediate step to generate the form, which allows the insert of instances but not the update of existing ones. An intuitive configuration interface is provided to the administrator user. The form generation takes this configuration as input, so the solution easy to evolve. The implementation of the approach requires to be upgraded.

The VocBench Custom Forms approach [10] is a data-driven form definition mechanism to create and visualize entities represented in a high-level abstraction language, the Lemon language, aimed to be a bridge between texts and RDF triples. This language is specifically designed to represent lexical-semantic resources while abstracting the technical complexities of semantic web languages like OWL and RDF. A key feature of Lemon is its ability to handle the reification required to bypass RDF's constraint that prevents literals from serving as subjects in RDF triples. While this functionality bears some similarity to OntoForms' approach of transparently creating intermediate instances (such as those used for representing ternary relations), there appears to be an important distinction: to our knowledge, Lemon's implementation focuses exclusively on enabling literals as triple subjects, without addressing the broader challenge of hiding intermediate nodes for both object and data properties.

Another approach is Fresnel [11], which defines a presentation-specific language through an extensible RDF vocabulary. Unlike OntoForms' focus on ontology population, Fresnel specializes in modelling content presentation knowledge, specifying both what content to display and how to style it. While both systems share an end-user orientation, Fresnel's scope is fundamentally different as it does not support ontology or knowledge graph population.

Our analysis examines the discussed approaches through two complementary lenses: their external usability features for different user types, and their internal implementation characteristics. For administrators handling ontology management and form configuration tasks, the configuration burden emerges as a critical differentiator between solutions. While some systems provide administrator-friendly interfaces, others impose more laborious setup processes. The VocBench Custom Forms approach exemplifies this challenge, despite offering a configuration interface, it demands administrators to master the Lemon language and manually map its elements to semantic web constructs, creating significant overhead. OntoForms addresses these configuration challenges through an interface where administrators can effortlessly hide irrelevant classes, particularly those serving as intermediate instances, and suppress properties not intended for direct instantiation.

For end users working with domain applications, the quality of interaction is driven by data manipulation capabilities. Solutions vary widely in this dimension, ranging from those limited to instance visualisation to implementations that support both instance creation and updating. A particularly challenging aspect is the transparent handling of the structural complexity of the ontology. Many ontologies contain intermediate classes to model n-ary relationships, elements that are crucial for formal correctness but irrelevant to end users. An optimal solution should automatically generate and manage these connective instances while keeping them hidden from users, presenting only domain-relevant information. VocBench Custom Forms takes a narrower approach, focusing specifically on enabling literals as RDF triple subjects. Some competing solutions venture into allowing end users to modify the ontology structure itself by adding classes or properties. OntoForms deliberately avoids this direction, maintaining that structural modifications to ontologies should remain the exclusive responsibility of ontology engineers, rather than domain application users. This philosophical distinction underscores our commitment to providing domain experts with tools specifically tailored for data interaction rather than ontology engineering.

From an implementation point of view, we mainly evaluate the maintainability of each solution. Several existing approaches, including those proposed by Gonçalves et al. and Rutesic et al. present significant maintenance problems. These systems require manual modifications of the internal components of the

user interface each time the underlying ontology evolves, which is undesirable for sustainable deployment. This automatic regeneration capability is a critical differentiator for maintainable systems, as it eliminates the need for error-prone manual intervention while ensuring synchronisation between interface and ontology.

Finally, we consider it highly relevant that form generation systems depend on their ability to incorporate ontological reasoning services. Relying on declared axioms alone would be insufficient, as the solution must also process inferred knowledge to properly capture all ontological constraints within the generated form structure. This reasoning capability plays three crucial roles: it allows the system to identify and enforce implicit relationships that aren't explicitly stated, to validate user input against inferred constraints that emerge from the logic of the ontology, and finally to present interfaces that are both complete and logically consistent. By simultaneously introducing automated reasoning capabilities and our approach to handling intermediate classes, we address two major limitations that persist in current solutions.

Table 1 presents a comparative analysis of the approaches based on the aforementioned features. The asterisk (∗) accompanying "Yes" indicates partial functionality with certain limitations. For instance, while the VocBench approach provides the core capability, its implementation requires administrators to learn the Lemon language, resulting in reduced user-friendliness. Similarly, for the "Data editing usability" feature, the Yes∗ means that the approach supports instance creation but lacks instance updating functionality.

Table 1
Comparison of approaches

Approach	User administration interface	Usability for data editing	Hide intermediate instances	Use of reasoning
Gonçalves et al.	No	Yes∗	No	No
Liu.	Yes	Yes∗	No	No
Vcelak et al.	Yes	Yes	No	No
Rutesic et al.	No	Yes∗	No	No
Peng at al.	not appl.	Yes	No	No
Rocchetti et al.	Yes	Yes∗	No	No
VocBench	Yes∗	Yes	Yes∗	No
Fresnel	Yes	No	No	No
Ontoforms	Yes	Yes	Yes	Yes

3. Overview of the OntoForms architecture

This section describes the general architecture of our solution, which leverages the OntoForms software component to enhance the maintainability of ontology-based applications. A key advantage of Onto-Forms is its ability to dynamically regenerate data entry interfaces in response to evolving application requirements and corresponding ontology updates, significantly reducing maintenance overhead.

As previously discussed, OntoForms serves two distinct user profiles: *Domain-expert* users who provide domain-specific data but lack ontology expertise. These users interact with automatically generated forms to populate and manage ontology instances without exposure to ontological complexities. *Administrator users* (domain experts with ontology design knowledge) who configure form generation settings and manage how forms adapt to the ontology structure.

Figure 1 illustrates the proposed architecture, by using the standard C4 model language [12]. The left side of the figure shows the OntoForms System which provides services to the administrator user and the domain application. The frontend application OntoForms Admin provides a user interface to interact with the OntoForms System, making use of different endpoints provided by the OntoForms Core API, which is the main component of the architecture. The main administration services provided by the OntoForms

Core API are ontology uploading and retrieval to and from the Triplestore, and the form configuration and generation.

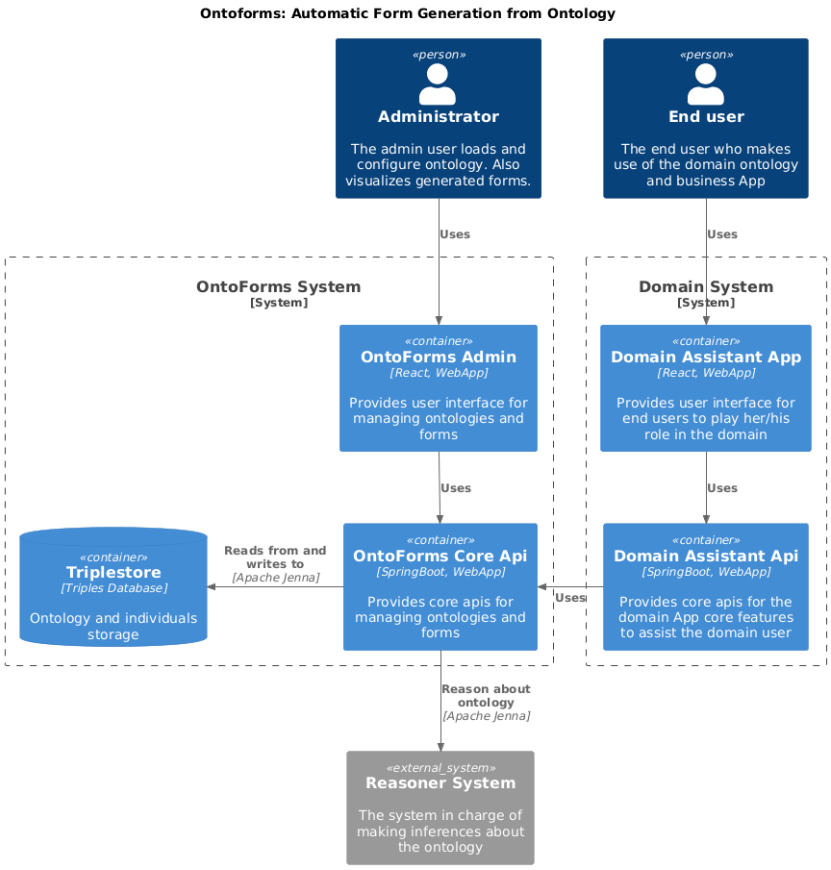

Figure 1: Architecture of the solution

The right-hand side of the Figure 1 shows the domain system operated by domain-expert end users. While domain applications can be implemented using various architectural styles, we propose a layered architecture for this complete solution, consisting of a front-end component (Domain Assistant App) and a back-end component (Domain Assistant API). As shown in Figure 1, the back-end uses endpoints provided by the OntoForms Core API to fulfil several requirements. In particular, the OntoForms Core API provides the structural framework that enables the front-end to render user interfaces directly from the domain ontology. Moreover, it provides endpoints for ontology population, handling both: domain-expert user-entered instances and transparent generation of intermediate instances required for the complete ontology population (while hiding this complexity from end users).

The main contribution of the present work is an ontology-driven interface generation algorithm (implemented in OntoForms Core API) that automatically produces the structure that describes the user interface to populate a class with its properties, given: (1) the ontology and (2) the target class. This algorithm is described in detail in the next section.

4. OntoForms Core API

The OntoForms Core API is the component that works on top of an ontology to generate the form structure for entering individuals in each class, starting from a class selected by the administrator user, called "main class". The structure generated by the OntoForms Core API will then correspond to the subgraph of the ontology, accessible from the "main class", by means of its declared properties as well as the properties inferred by the reasoner. The instantiation of this class, called the "main class", involves filling in all data

properties and object properties associated with the individual that populates the class. Then, the form structure generated by the OntoForms Core API will correspond to the ontology's subgraph accessible from the main class", through its declared properties as well as the properties inferred by the reasoner. This processing is depth-first, similar to traversing a spanning tree in the ontology graph. Algorithm 1 illustrates the ontology processing performed by the OntoForms Core API. The algorithm initiates with the main class and subsequently constructs one or more sections that organize a collection of fields. The components defined in Algorithm 1 are as follows:

- $\mathscr{O}$: Represents the domain ontology.
- $\mathscr{O}^{\models}$: Denotes the deductive closure of O.
- C: Refers to the main class.
- **SubC** $= \{SC_1, \ldots SC_n\}$: Represents the set of subclasses derived from C.
- **Props** $= \{p_1, \ldots p_m\}$: Constitutes the set of object and data properties associated with the domain of C or any superclass of C.
- $Range(p)$: Indicates the range of the property p.
- *Form*: Describes the structure of the form generated for $\mathscr{O}^{\models}$.
- *Section*: Defines the structure that organizes fields and subsections within the Form.

Algorithm 1 OntoForms

Input: $\mathscr{O}^{\models}$, C
Output: *Form*
  SubC ← obtain subclasses of C
  if SubC is empty **then**
    Props ← obtain properties with domain C
    for all p_i in **Props** $(1 \leqslant i \leqslant m)$ **do**
      if p_i is not calculated **then**
        if p_i is a data property **then**
          Section ← add a new field
        end if
        if p_i is an object property **then**
          if $Range(p_i)$ is not empty and p_i is transparent **then**
            Section ← add OntoForms($\mathscr{O}^{\models}$, $Range(p_i)$)
          else
            Section ← add a new field
          end if
        end if
      end if
    end for
    Form ← add *Section*
  else
    for all SC_i in **SubC** $(1 \leqslant i \leqslant n)$ **do**
      Form ← add OntoForms($\mathscr{O}^{\models}$, SC_i)
    end for
  end if
  return *Form*

Algorithm 1 is executed recursively by setting each subclass as a parent class until the most specific subclass is reached. This subclass corresponds to a new section in the form, which will contain a set of fields for entering the values of the data and object properties associated with this class, within the domain of the properties.

The high-level description of the algorithm is as follows:

- For each data property, a new field in the form is created to enter or select the corresponding individual.
- For each object property, a new field is created in the form to select one or more existing preloaded individuals. Here, a selection box is open in the corresponding section. If the object property is functional, the box will only allow a single selection. Otherwise, it will allow multiple individuals to be selected.
- If the Range class of the Object property has been configured as "transparent", i.e. not relevant to the user, instead of creating a new field in the section, the algorithm is executed recursively by setting the Range class of the Object property as the Main class. In this case, a new instance of the Range class is automatically created and linked to the individual of the main class through the Object property. This allows users to set values for properties whose scope is within range class.

Note that if the main class is part of a complex expression in the object property domain that uses the conjunction operator with another class, then the object properties will be ignored. The reason for this is that when the domain is defined as a conjunction of classes, there may be individuals of any of the classes that are not members of the restricted domain. If individuals were to be specifically included in the domain of this relationship, a class defined as equivalent to the expression of that domain should be placed as the main class.

Note that OntoForms Core API takes the names of the form sections and fields from existing annotations and names in the ontology. However, these names can be customized.

4.1. Implementation Issues

The OntoForms Core API is developed using Java-based technologies, specifically leveraging: Spring Boot 3 framework for backend infrastructure and Java 17 as the runtime environment.

As illustrated in Figure 2, the system follows a layered architectural pattern comprising three principal components:

- API Layer: Handles request/response cycles and endpoint management
- Service Layer: Contains core business logic and algorithm implementation
- Persistence Layer: Manages data access and storage operations

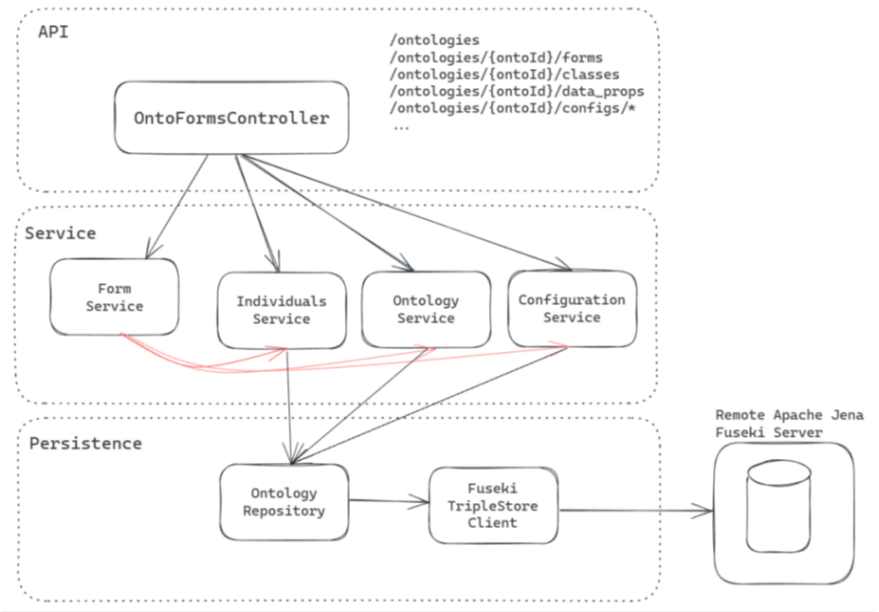

Figure 2: OntoForms Core API structure.

One of the challenges of ontology processing is retrieving the set of object property relationships defined and inferred for a given class. Retrieving these using SPARQL involves writing queries with complex logic. As described in [13], one consequence of the RDF design is that, because of the open-world paradigm, it is not possible to answer the question "What properties can be applied to the resources of class X? Strictly speaking, the RDF answer is "any property". But to answer the question of which properties have a particular class in their domain, one must also take into account properties that do not have their domain defined, and properties that have the closure of the parent classes in their domain on the hierarchy of classes of the given class.

In this work, to obtain the properties of a given class, we use a mechanism built into Jenna that supports a frame-like view of resources using the reasoner's capabilities. It defines properties of a class as only those properties that don't add any new information when applied to resources already known to be of that class. These properties are the relationships that need to be filled in on a new individual form of the main class. For this purpose, we used Apache Jena SDK to load an ontology model with one of the configurable reasoners (we use the OWL_DL_MEM_TRANS_INF reasoner configuration), and then retrieve from the target OntClass the declaredProperties filtered with filterKeep of data and object properties.

4.2. Limitations and evaluations

While description logic reasoning services offer significant advantages (as previously discussed), they also present computational performance challenges. The complexity for $\mathcal{SROIQ}$ is N2ExpTime-complete, which necessitates careful design considerations. Consequently, OntoForms is specifically optimized for generating application interfaces in well-defined, concrete domains represented by small to medium-scale ontologies.

Another limitation of the tool is that it is not possible to create object properties whose range is an instance that does not yet exist. In such cases, it is necessary to first create that instance.

We are currently working on healthcare applications where the user is a specialist physician who uses the application during patient consultations. In this context, the doctor inputs the patient's data instance with the necessary properties, so that the reasoner can apply constraints and make inferences to provide a diagnosis or recommend disease prevention procedures based on the patient's data. An example is a breast cancer prevention system [14], which invokes the OntoForms Core API to generate a form for the doctor to enter the patient data required for the reasoner to determine preventive imaging studies as specified in the recommendation guidelines from recognized health institutions. This application has been validated by a board-certified breast cancer imaging specialist.

5. People ontology example

This section describes how the OntoForms Core API is applied to the People ontology to generate the user interface form structure. We present the OntoForms Core API administrator web page and preview the generated form, designed for adding a new individual to a selected main class.

In this case, we select the *Person* class as the main class. To generate a form for populating the ontology, the first step is to upload the ontology to the OntoForms system through the administrator website (accessible via http://ontoforms-admin-ui.web.elasticloud.uy/). Once the ontology is uploaded, it can be explored by selecting the 'Onto Detail' section, as shown in Figure 3. This allows users to review its classes, hierarchy, properties, and individuals. The administrator can then choose an ontology and a main class from the hierarchy tree to generate the default form for the People ontology and the *Person* class, displaying all properties for user input.

Once the default form is visualized, the administrator user can introduce different configurations. As was mentioned above, a contribution of OntoForms is the possibility of configuring intermediate classes as transparent, by hiding them in the form. In this case, the class *DirectAncestry* is added to play the role of the intermediate class that connects each instance of *Person* to two instances via the properties

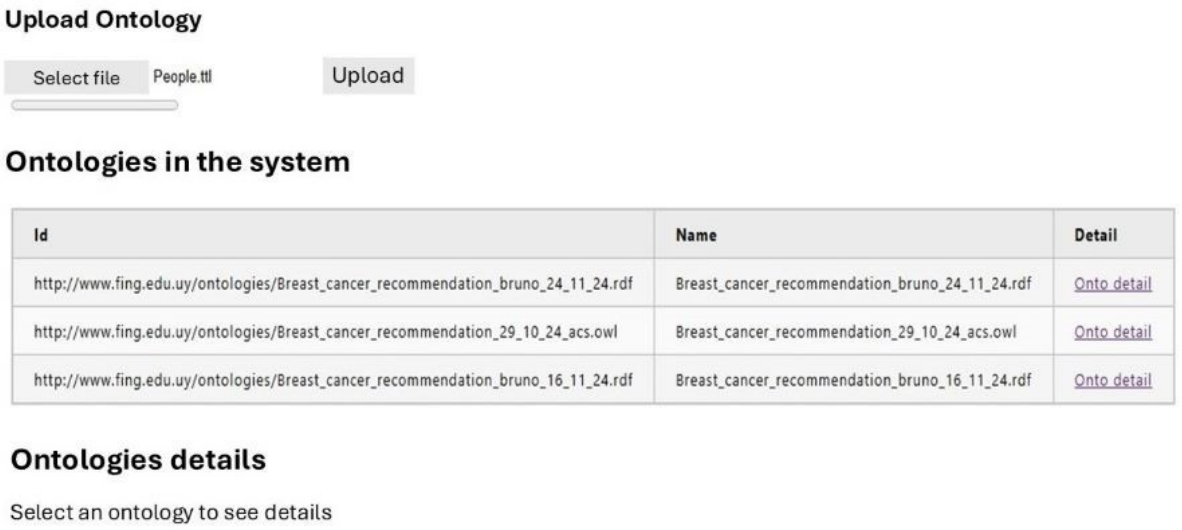

Figure 3: Explore uploaded Ontology

father and *mother*. For this, a new property *has_direct_ancestry* is created is created with domain *Person* and range *DirectAncestry* is created, and ranges of properties *father* and *mother* are set with the class *DirectAncestry*. When the administrator user configures the class *DirectAncestry* as transparent, instead of displaying a field for the property *has_direct_ancestry* to be entered, a new section "Direct Ancestry" is rendered with two fields to fill out the values of properties *father* and *mother*. Figure 4 illustrates the example. Moreover, it is possible hidding properties that are not intended to be instantiated.

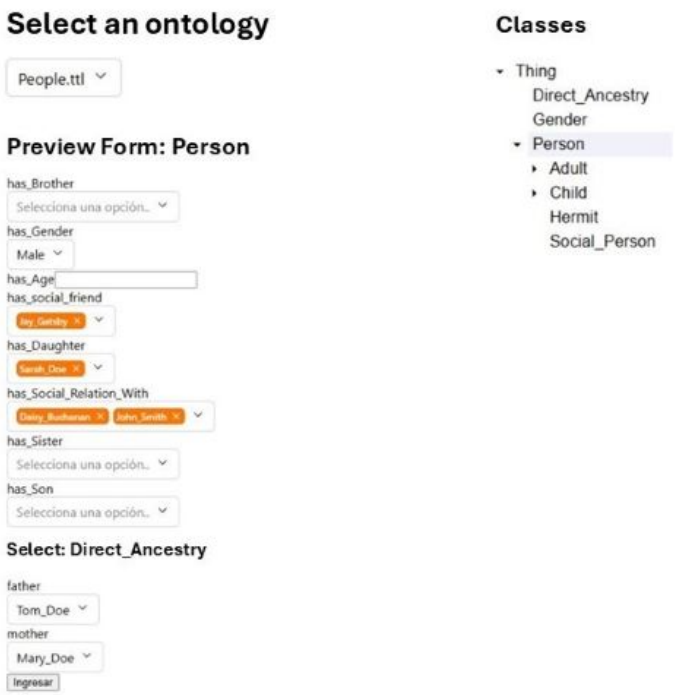

Figure 4: Form with the class DirectAncestry hidden

The other relevant contribution of OntoForms is the use of reasoning services to consider entailments

in the generated form. For example, properties *father* and *mother* have range *hasGender value Male* and *hasGender value Female* respectively. Since there is no person in the ontology explicitly declared as an instance of *hasGender value Male* and *hasGender value Female*, it would not be possible to present instances for the user to select. By executing the reasoner, OntoForms can present the user instances of *Person* that meet these restrictions. Figure 5 illustrates the example. On the left side, for the property *father*, we can see that only *TomDoe* can be selected, and on the right side the corresponding query is showed to verify that only *TomDoe* meets *hasGender value Male*. The same happens for the property *mother*. Similarly, to obtain the subclasses of the main class and the properties whose domain is the main class, Algorithm 1 considers both the declared axioms and the axioms inferred by the reasoning services.

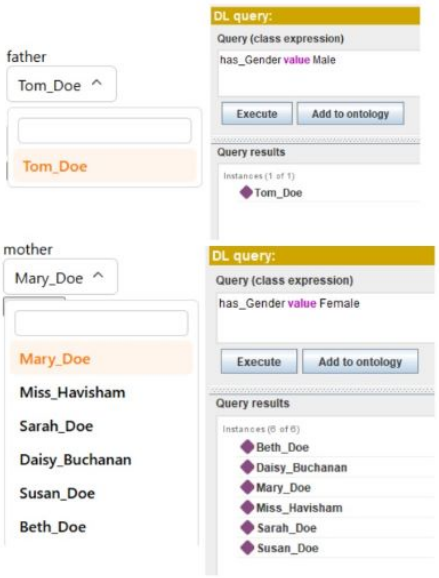

Figure 5: Example of inference Father-Mother.

6. Conclusions and future work

This work introduces OntoForms, a software component that takes a domain ontology as input and generates a structured description of a user interface. This approach enhances the maintainability of ontology-based applications by allowing them to utilize OntoForms endpoints for dynamic domain-expert user interface generation.

The core of OntoForms is an algorithm that, given an ontology and a class within it, generates a structure representing a form to insert or update an individual of this class. It also instantiates the object and data properties associated with this individual. The key distinguishing feature of OntoForms is its use of reasoning services based on the formal semantics of description logic, ensuring that both declared and entailed axioms are used to build the user interface. This use of the reasoner benefits the domain user in the sense that the user interface has more meaning and is more tailored to the domain. For example, when rendering a field with a select option generated from an object property with the range class being an expression, then the reasoner classifies the set of instances that belong to the range class, to be presented to the user. If no reasoner were used, then no evaluation of the expression could be done and no eligible candidates could be shown.

OntoForms can be used by simply providing the ontology and the class to be instantiated. It also offers a configuration interface that allows the administrator to select which ontology classes and properties should be included or hidden in the form, according to domain application requirements.

An ontology-based application in the health domain was implemented to leverage the benefits of OntoForms. The usability of the application was validated by a health expert user. The next step is that

more health experts validate the usability of domain applications that use OntoForms.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] S. Staab, R. Studer, Handbook on Ontologies, 2nd ed., Springer Publishing Company, Incorporated, 2009.

[2] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS Q. 28 (2004).

[3] K. Conboy, R. Gleasure, E. Cullina, Agile design science research, in: B. Donnellan, M. Helfert, J. Kenneally, D. VanderMeer, M. Rothenberger, R. Winter (Eds.), New Horizons in Design Science: Broadening the Research Agenda, Lecture Notes in Computer Science, 10th International Conference, DESRIST 2015, Dublin, Ireland, May 20–22, 2015, Proceedings, Springer International Publishing, Cham, 2015, pp. 168–180.

[4] R. S. Gonçalves, S. W. Tu, C. I. Nyulas, M. J. Tierney, M. A. Musen, An ontology-driven tool for structured data acquisition using Web forms, Journal of Biomedical Semantics 8 (2017).

[5] F. Liu, An ontology-based approach to automatic generation of gui for data entry. Master's Thesis, 2009. URL: https://scholarworks.uno.edu/td/1094.

[6] P. Vcelak, M. Kryl, M. Kratochvil, J. Kleckova, Ontology-based web forms — acquisition and modification of structured data, in: 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2017.

[7] P. Rutesic, M. Radonjic-Simic, D. Pfisterer, An enhanced meta-model to generate web forms for ontology population, in: Knowledge Graphs and Semantic Web - Third Iberoamerican Conference and Second Indo-American Conference, KGSWC 2021, Kingsville, Texas, USA, November 22-24, 2021, Proceedings, 2021.

[8] Y. Peng, Y. Kang, Ontology-based dynamic forms for manufacturing capability information collection, in: IFIP Advances in Information and Communication Technology, 2013.

[9] N. Rocchetti Martínez, G. Labandera, Generación automática de formularios para el ingreso de datos en ontologías - aplicación en la implementación de una ontología de percepciones de piezas de arte. Degree Thesis, 2016.

[10] M. Fiorelli, T. Lorenzetti, M. T. Pazienza, A. Stellato, Assessing vocbench custom forms in supporting editing of lemon datasets, in: Language, Data, and Knowledge - First International Conference, LDK 2017, Galway, Ireland, June 19-20, 2017, Proceedings, 2017.

[11] E. Pietriga, C. Bizer, D. R. Karger, R. Lee, Fresnel: A browser-independent presentation vocabulary for RDF, in: The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings, 2006.

[12] Standard C4 model, https://c4model.com/, Last date accessed June 2025.

[13] Standard Apache Jenna - RDF frames, https://jena.apache.org/documentation/notes/rdf-frames.html, Last date accessed July 2024.

[14] B. Szilagyi, E. Rohrer, Y. Anchén, R. Motz, An ontological approach to breast cancer screening: Risk assessment and personalized testing recommendations, in: Companion Proceedings of the 43rd International Conference on Conceptual Modeling: ER Forum, Special Topics, Posters and Demos Co-located with ER 2024, Pittsburgh, Pennsylvania, USA, October 28-31, 2024, 2024.