MASEO: A Multi-Agent System for Explainable Ontology Generation*

Jiayi Li[1,*,†], Ziyuan Wang[1,†], Daniel Garijo[1,†] and María Poveda-Villalón[1,†]

[1]Ontology Engineering Group, Universidad Politécnica de Madrid, Boadilla del Monte, 28660, Spain

Abstract

Ontology development remains a complex and labor-intensive task that often requires substantial expertise to ensure competency question coverage, logical consistency, and adherence to modeling best practices. In this paper, we present MASEO, a multi-agent framework for automated ontology generation from a set of competency questions. Inspired by approaches like NeOn-GPT, MASEO is organized in four stages to produce valid OWL ontologies: an Ontology Generation Agent for initial ontology construction, a Syntax Repair Agent leveraging RDFLib for structural error correction, a Logical Consistency Agent integrating the HermiT reasoner, and an Ontology Pitfall Agent addressing modeling issues identified by the OOPS! scanner. A novel feature of our approach is a provenance tracking mechanism embedded in the resultant ontology, capturing the rationale behind the addition of each term and axiom. This makes the ontology development process more transparent and traceable, addressing an important limitation of existing ontology generation pipelines. We demonstrate MASEO through three ontology engineering case studies of varying requirement scale and language: an Infrastructure Ontology, a Vehicle Census Ontology, and a Video Game Ontology. Across these cases, CQ coverage reaches 100%, 60.7%, and 54.4%. We further report concept label matching results under exact, lexical, and semantic strategies, showing that even when exact label matches are limited, lexical and semantic methods recover a substantial proportion of conceptual overlap.

Keywords

Ontology Engineering, Automated Ontology Generation, Large Language Models, Multi-Agent Systems

1. Introduction

Ontology development is a core component of ontology engineering. Established methodologies such as LOT [1], NeOn [2] and the "Ontology Development 101" guide [3] provide guidance for ontology engineers, covering activities that span from requirements specification [4, 5, 6] to ontology implementation [7, 8], publication [9, 10], and evaluation [11, 12, 13]. However, methodology adoption often requires substantial expertise and time [1], which limits scalability in dynamic and knowledge-intensive domains.

Furthermore, manual organization and verification of ontology elements remain laborious and error-prone tasks [14]. Ensuring syntactic correctness, logical consistency, and modeling quality typically requires iterative human intervention and the coordinated use of multiple validation tools [15, 16]. These challenges motivate the need for automation mechanisms within ontology development workflows. Recent advances in Large Language Models (LLMs) have demonstrated strong capabilities in knowledge extraction and structured reasoning from natural language inputs [17, 18, 19, 20]. Initial efforts in this area reported promising results on using generative AI for foundational tasks, such as the automated creation of competency questions (CQs) [21, 22, 23] and initial conceptual modeling [24, 25, 26], sparking growing interest in the application of LLMs to broader ontology engineering tasks.

Among these efforts, systems have emerged that combine LLM generation with external validation tools to support more automated ontology workflows. For instance, NeOn-GPT [27] demonstrated the

LLM4KGOE '26: 1st Workshop on LLM-driven Knowledge Graph and Ontology Engineering, May 10–14, 2026, Dubrovnik, Croatia

*Corresponding author.

†These authors contributed equally.

li.jiayi@upm.es (J. Li); ziyuan.wang@upm.es (Z. Wang); daniel.garijo@upm.es (D. Garijo); mpoveda@fi.upm.es (M. Poveda-Villalón)

0009-0001-9475-8159 (J. Li); 0009-0000-6228-4713 (Z. Wang); 0000-0003-0454-7145 (D. Garijo); 0000-0003-3587-0367 (M. Poveda-Villalón)

feasibility of manually combining LLM reasoning with ontology validation tools to detect and repair syntax errors and modeling issues. However, two important limitations remain. First, existing pipelines operate as linear prompt sequences without dedicated agents for individual repair tasks: none of the existing approaches provides a fully integrated end-to-end pipeline spanning generation, validation, and automated repair within a single coherent framework. Second, and more critically, current systems produce OWL ontology files without documenting the rationale behind individual modeling decisions, as no record is kept of which competency question (CQ) [28] motivated the introduction of a concept or property, or which validation feedback triggered a corrective change. As a result, ontology engineers cannot trace why specific terms were introduced or modified during the development process. To address these limitations, this work proposes MASEO, a fully automated ontology generation and refinement pipeline inspired by the NeOn-GPT framework [27]. Our approach generates ontologies directly from CQs and refines them through a multi-agent validation workflow integrating external tools, including an RDFlib syntax validator, the HermiT[1] reasoner [29], and the OOPS![2] pitfall scanner [30]. In MASEO, each agent records its actions directly within each ontology term as annotations, generating a human readable provenance trace embedded in the final ontology. The complete framework, system prompts, and experimental datasets are available on GitHub[3] and archived on Zenodo [31].

The main contributions of this work include:

- MASEO, a fully automated end-to-end pipeline based on a multi-agent architecture that coordinates four role-based agents to transform CQs directly into formal OWL ontologies. The framework is designed to accommodate additional agents for other ontology engineering tasks, such as ontology reuse.
- A provenance reporting mechanism that automatically produces a structured record linking each ontology term and axiom to the CQ or validation feedback that motivated its introduction, making the history of each entity transparent and traceable. This mechanism allows ontology engineers to understand the rationale behind each modeling decision, making it easier to review, adapt, and maintain the final ontology.
- Three case studies of varying scale (5, 28, and 68 CQs) and two languages (English and Spanish), covering the domains of infrastructure management, vehicle registration, and video games. Two of the ontologies were developed by ontology engineers from scratch, while the third one (video games) is part of the CORAL [32] corpus.

The remainder of this paper is structured as follows. Section 2 reviews related work on LLM-based ontology engineering. Section 3 presents our multi-agent MASEO framework. Section 4 describes the experimental setup and reports the evaluation results for our three case studies. Section 5 discusses the findings and limitations of our study, and Section 6 concludes the paper. Appendix A provides additional details on the datasets and prompting templates used in our experiments.

2. Related Work

The integration of Large Language Models (LLMs) into ontology engineering has evolved from supporting isolated tasks to enabling more integrated workflows, as highlighted in recent surveys [33, 34].

Early research primarily positioned LLMs as assistants for individual engineering activities. Studies in this category investigated the extraction of concepts and axioms from natural language inputs for ontology conceptualization and encoding [35, 36, 37, 38]. Other work explored tasks such as CQ specification and SPARQL query construction to support ontology evaluation and knowledge access [25, 39]. Furthermore, several studies investigate the use of LLMs for ontology validation and error detection. For instance, Tsaneva et al. [11] employ GPT-4 to perform ontology restriction

[1] https://www.hermit-reasoner.com/
[2] https://oops.linkeddata.es/
[3] https://github.com/oeg-upm/maseo

verification, while other approaches focus on detecting modeling pitfalls and taxonomic errors [12, 25]. While these methods demonstrate the generative and analytical capabilities of LLMs, they typically address isolated stages rather than supporting a cohesive engineering lifecycle.

Building on these developments, recent research investigates workflow-oriented designs that integrate LLM capabilities into ontology engineering pipelines. Val-Calvo et al. [40] with OntoGenix, Lippolis et al. [41] with Ontogenia, and Zhang et al. [13] with OntoChat exemplify this generation and validation paradigm. In specialized domains, Ouédraogo et al. [42] demonstrate a Text-to-OWL approach for constructing ontologies to improve medical recommendations. A notable example in this category is NeOn-GPT [27], which combines LLM-based generation with formal validation tools such as RDFLib, HermiT, and OOPS!. However, as an application integrated into the Metaphactory platform [43], it remains semi-automated: transitions between stages must be manually triggered by engineers, and no mechanism is provided to document the rationale behind individual modeling decisions. Moving toward more structured agentic behavior, Ivanova [44] proposes an architecture for step-by-step ontology learning with explicit role decomposition within the workflow.

Despite these advances, two important limitations remain. First, existing approaches typically focus on specific stages of ontology engineering and rarely provide a fully automated end-to-end workflow for ontology generation, validation, and repair [33]. Second, and more critically, these frameworks generally do not document the rationale for specific modeling decisions, which is critical for transparency and maintenance [45, 46]. Addressing these gaps requires unified approaches that preserve both automation and transparency throughout the refinement process.

3. MASEO: Multi Agent System for Explainable Ontology Generation

In this section, we introduce our framework. We first present the structured ontology representation that serves as the shared state across all agent stages. Next, we describe each pre-defined agent in detail, explaining how they read and modify this shared representation. Finally, we show a complete step-by-step example to illustrate how provenance information is transparently recorded at each stage of the framework.

3.1. Ontology Representation

Table 1 shows an overview of the shared representation for agent state exchange in our framework. Whenever an agent introduces or modifies an ontology entity, the corresponding changes are first recorded in the entity's structured fields and subsequently serialized into RDF/XML. In this way, the evolution of the ontology remains tightly coupled with the provenance information describing how and why each modeling decision was made. Most of the fields of this structured record correspond directly to standard OWL/RDF constructs. The `rdf:type` field specifies whether the entity represents a class, object property, or datatype property. The `rdfs:label` and `rdfs:comment` fields provide the human-readable name and description of the entity. For properties, the `rdfs:domain` and `rdfs:range` fields capture the domain and value constraints of a given property. For classes, the `rdfs:subClassOf` field records superclass relations or class restrictions that are explicitly materialized in the generated ontology ($Onto_{\text{gen}}$).

In addition to these structural fields, the representation incorporates dedicated provenance fields. The `vaem:rationale` field maintains an append-only, agent-attributed log of the justifications behind each modification applied during successive refinement iterations. This property is adopted from the Vocabulary for Attaching Essential Metadata (VAEM),[4] as it provides a semantically appropriate mechanism for capturing the rationale behind ontology modeling decisions. The `dc:source` field records the CQs, detected pitfalls, or reasoner feedback that motivated each entity creation or revision step. This property is adopted from the Dublin Core Metadata Initiative (DCMI)[5], which is widely used

[4] https://linkedmodel.org/doc/vaem/1.2/
[5] https://www.dublincore.org/

to attribute resources to their originating sources.

Finally, logical axioms associated with an entity are also preserved. In the current implementation, complex axioms beyond `rdfs:subClassOf` and `owl:disjointWith` (e.g., universal, existential or cardinality constraints) are not materialized as formal OWL statements but are instead recorded as annotations within the entity representation. This design choice allows the framework to retain the modeling decisions suggested during the generation and validation stages while avoiding the introduction of additional logical constraints that may affect ontology consistency during intermediate refinement steps.

Together, the structural and provenance fields embedded in this representation ensure that every ontology term in the generated OWL file carries a complete record of its origin. This record links each term directly to the agent action and input evidence (e.g., CQs or validation feedback) that motivated its introduction or modification.

Table 1
Structured representation of ontology entities used across the multi-agent pipeline, including the main fields, their definitions, and example values derived from competency questions such as "What is the username of a player in a Game?"

Field	Definition	Example value
Type (`rdf:type`)	Indicates whether the entity is an `owl:Class`, `owl:ObjectProperty`, or `owl:DatatypeProperty`.	`:Player rdf:type owl:Class; :hasUsername rdf:type owl:DatatypeProperty`
Label (`rdfs:label`)	Provides a human-readable name for the entity.	`"Player"; "has username"`
Comment (`rdfs:comment`)	Provides a textual description of the meaning of the entity.	`"A person who plays games."; "Relates a player to the player's username."`
Rationale (`vaem:rationale`)	Records the justification for entity creation or modification across refinement iterations.	`"Derived from CQ [number] about the username of a player."`
Source (`dc:Source`)	Records the CQ or validation feedback from which the entity or revision was derived.	`"What is the username of the player in this game?"`
Subclass of (`rdfs:subClassOf`)	(Classes only) Records subclass relations or logical restrictions involving the class.	(derived from the CQ set, not from the sample CQ) `:Player rdfs:SubClassOf :Human`
Domain (`rdfs:domain`)	(Properties only) Specifies the class to which a property applies.	`:hasUsername rdfs:domain :Player`
Range (`rdfs:range`)	(Properties only) Specifies the value type or class associated with a property.	`:hasUsername rdfs:range xsd:string`
Disjointness (`owl:disjointWith`)	(Classes only) Declares that two classes are mutually exclusive, meaning that no individual can belong to both classes at the same time.	(derived from the CQ set, not from the sample CQ) `:Player owl:disjointWith :Game`
Other Axioms	Captures logical constraints as structured XML comments to preserve modeling intent.	`<!-- Axiom: Player has exactly 1 username -->` (captured as comments)

3.2. Multi-Agent Explainable Ontology Generation Framework

The framework consists of four sequential stages, as illustrated in Figure 1: (1) ontology generation, (2) syntax repair, (3) logical consistency checking, and (4) pitfall resolution. Each stage is supported by a dedicated agent defined at creation time by a stage-specific natural language instruction. Appendix A provides a summary of the agents, their roles, and the prompt information they received (see Table 5 for a summary) and representative instruction examples.

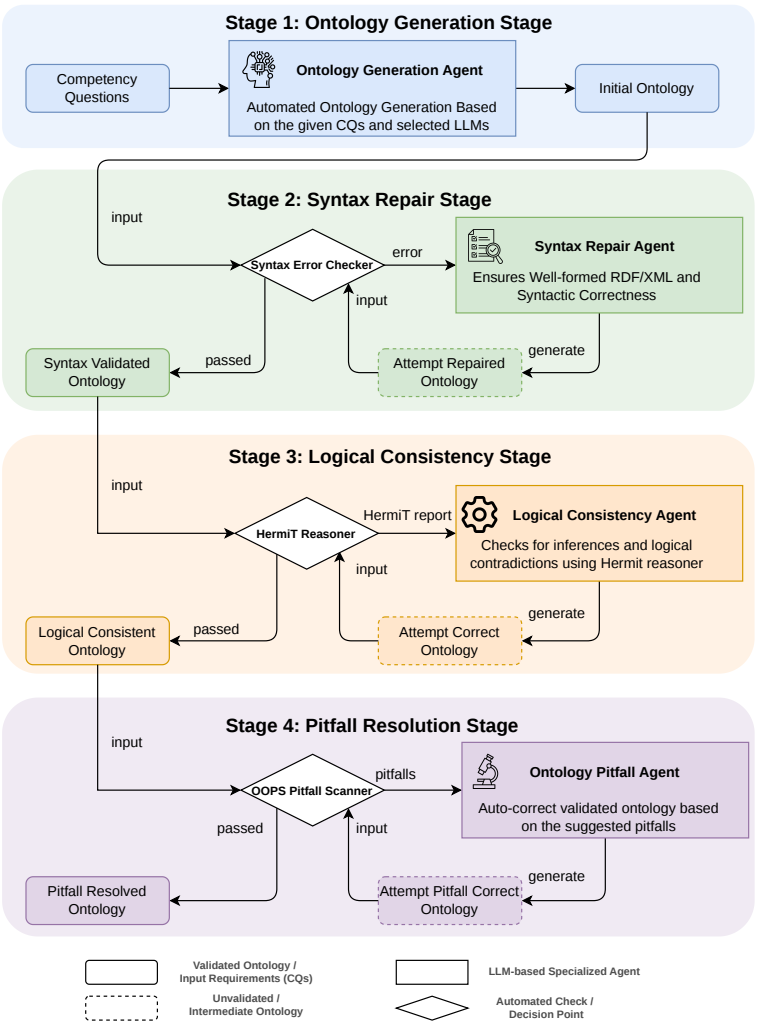


Figure 1: The MASEO framework consists of four specialized LLM agents (Generation, Syntax Repair, Logical Consistency, and Pitfall Resolution) through iterative validation loops. Solid borders denote validated data states and final outputs, while dashed borders represent intermediate, non-validated candidate ontologies generated during the repair process.

Stage 1: Ontology Generation. The first stage transforms the input CQs into an initial OWL ontology. The Ontology Generation Agent is defined as an experienced knowledge engineer modeling a specific domain. During execution, it receives the complete set of CQs as input and generates an initial ontology in RDF/XML. For each entity introduced, the agent populates the fields `Type`, `Label`, `Comment`, and `Source`. The agent also identifies taxonomic relationships and defines the `rdfs:subClassOf` hierarchy for relevant classes. For properties, the agent additionally specifies `Domain` and `Range`. The creation rationale for each term, indicating which specific CQ motivated its inclusion, is recorded in `vaem:rationale`.

Stage 2: Syntax Repair. After generation, the ontology is validated using a syntax checker. If syntax errors are detected, the predefined Syntax Repair Agent is invoked with the incorrect ontology and the corresponding RDFLib error message as input. The agent repairs only structural or serialization errors in the RDF/XML representation and does not modify provenance fields such as `dc:source` or `vaem:rationale`. This process iterates until the syntax checker reports no errors.

Stage 3: Logical Consistency. Once syntactically valid, the framework invokes the HermiT OWL reasoner to perform logical consistency checking and subsumption reasoning. If inconsistencies are detected, the predefined Logical Consistency Agent is invoked with the ontology and the HermiT error

report as input. For each modification, the agent appends a new entry to `vaem:rationale` and updates `dc:source`. Any revision triggers a backtracking loop to Stage 2 for syntax re-validation before the reasoner is re-run to confirm consistency.

Stage 4: Pitfall Resolution. In the final stage, the ontology is analyzed using the OOPS! scanner against a taxonomy of 41 common design pitfalls. The resulting XML report is distilled into a textual summary of the form *Pitfall Name* : *Pitfall Description*, which serves as contextual feedback for the agent. The predefined Ontology Pitfall Agent treats the OOPS! report as a set of debugging suggestions rather than strict errors, allowing for ontological judgment when deciding whether a flagged pitfall requires structural modification. For each new or modified entity, the agent records the corresponding pitfall identifier and description in the `vaem:rationale` field and updates `dc:source`.

Following each pitfall-repair iteration, the revised ontology undergoes a comprehensive re-validation waterfall (re-running Stage 2 and Stage 3) to ensure it remains syntactically and logically sound. The workflow finishes when OOPS! reports no remaining pitfalls and the ontology passes the final syntax and consistency checks. To avoid infinite loops, a maximum of five tries are attempted for each step.

3.3. Provenance Tracking: A Step-by-Step Example

To illustrate how the structured representation is populated and updated across all four agent stages in the MASEO framework, we trace the entity `Player` through an example drawn from the Video Game Ontology domain. Note also that at each stage, if the validator reports no issues, the corresponding agent is not invoked, and the pipeline proceeds to the next stage without modifying any entity record.

In this example, the relevant CQs are shown below:

Listing 1: Competency questions grounding the example

```
CQ2: Who are the friends of the player?
CQ6: Who does the player play with?
CQ25: What games has the player played?
```

Stage 1: Generation. The Ontology Generation Agent creates the class `Player` from CQ2, CQ6, and CQ25, and also introduces the object property `playsGame`. From CQ25, the agent further infers that `Player` and `Game` are fundamentally distinct entity types, a player *plays* games, and therefore cannot itself be a game, and correctly asserts an `owl:disjointWith` axiom between the two classes. However, the agent also erroneously introduces an `rdfs:subClassOf` axiom asserting `Player` as a subclass of `Game`, a hallucination unsupported by any CQs. This contradiction renders `Player` unsatisfiable. At this stage, the corresponding structured records are initialized with provenance metadata through `dc:source` and `vaem:rationale`.

Listing 2: Stage 1: RDF snippet with the initial generation containing a semantic contradiction

```
<owl:Class rdf:about="http://www.semanticweb.org/myontology#Player">
  <rdfs:label>Player</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <owl:disjointWith rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <dc:source>CQ2, CQ6, CQ25</dc:source>
  <vaem:rationale>
    [Ontology Generation Agent] Created class Player from competency questions.
  </vaem:rationale>
  <!-- Axiom: Player has exactly 1 username -->
</owl:Class>

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/myontology#playsGame">
  <rdfs:label>plays game</rdfs:label>
  <rdfs:domain rdf:resource="http://www.semanticweb.org/myontology#Player"/>
  <rdfs:range rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <dc:source>CQ25</dc:source>
  <vaem:rationale>
```

```
    [Ontology Generation Agent] Created property playsGame from competency questions.
  </vaem:rationale>
</owl:ObjectProperty>
```

Stage 2: Syntax Repair. RDFLib confirms that the Stage 1 output is well-formed RDF/XML, so no syntax repair is required, and the records remain unchanged.

Stage 3: Logical Consistency. HermiT identifies `Player` as unsatisfiable because the Stage 1 output conflict asserts `rdfs:subClassOf Game` and `owl:disjointWith Game`. The Logical Consistency Agent resolves this conflict by removing the `rdfs:subClassOf` axiom, which lacks grounding in any CQ, while preserving the `owl:disjointWith` axiom, which is semantically motivated by CQ25.

Listing 3: Stage 3: Consistency repair of `Player`

```
<owl:Class rdf:about="http://www.semanticweb.org/myontology#Player">
  <rdfs:label>Player</rdfs:label>
  <owl:disjointWith rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <dc:source>CQ2, CQ6, CQ25; HermiT: unsatisfiable class</dc:source>
  <vaem:rationale>
    [Ontology Generation Agent] Created class Player from competency questions.
    [Logical Consistency Agent] Removed rdfs:subClassOf axiom to restore satisfiability.
  </vaem:rationale>
</owl:Class>
```

Stage 4: Pitfall Resolution. OOPS! then detects that the object property `playsGame` is missing inverseOf axioms (P13). The Ontology Pitfall Agent addresses this by creating `isPlayedBy` and asserting the corresponding `owl:inverseOf` axioms. The provenance annotations are updated to record this pitfall-driven refinement, after which the ontology is re-validated through Stages 2 and 3 to ensure that no new syntax or consistency errors have been introduced.

As shown in Listing 4, the final RDF/XML record reflects contributions from all four stages: initial class and property creation at Stage 1, silent passage through Stage 2, a consistency fix of the `Player` class at Stage 3 through the drop of the conflict `rdfs:subClassOf` axiom while preserving the CQ-motivated `owl:disjointWith` axiom, and a pitfall-driven enrichment of the relation model at Stage 4 through the addition of an explicit inverse property. Because these records are embedded directly within the generated OWL file as `dc:source` and `vaem:rationale` annotations, ontology engineers can inspect each term and trace both the originating requirement and the validation feedback that subsequently refined it.

Listing 4: Final integrated RDF/XML snippet with provenance (Game class omitted for simplicity)

```
<owl:Class rdf:about="http://www.semanticweb.org/myontology#Player">
  <rdfs:label>Player</rdfs:label>
  <owl:disjointWith rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <dc:source>CQ2, CQ6, CQ25; HermiT: unsatisfiable class</dc:source>
  <vaem:rationale>
    [Ontology Generation Agent] Created class Player from CQ2, CQ6, CQ25.
    Asserted owl:disjointWith Game based on CQ25.
    [Logical Consistency Agent] Removed rdfs:subClassOf axiom to restore satisfiability.
  </vaem:rationale>
  <!-- Axiom: Player has exactly 1 username -->
</owl:Class>

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/myontology#playsGame">
  <rdfs:label>plays game</rdfs:label>
  <rdfs:domain rdf:resource="http://www.semanticweb.org/myontology#Player"/>
  <rdfs:range rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <owl:inverseOf rdf:resource="http://www.semanticweb.org/myontology#isPlayedBy"/>
  <dc:source>CQ25; OOPS!: missing inverse relation</dc:source>
  <vaem:rationale>
    [Ontology Generation Agent] Created property playsGame from CQ25.
```

Table 2
Overview of the CQ sets used in the evaluation, including their source ontology, scale, number of Competency Questions (CQs), application domains, and languages.

CQ Set Source	Size	#CQs	Application Domain	Language
Infrastructure Ontology	Small	5	Infrastructure Management	Spanish
Vehicle Census Ontology	Medium	28	Vehicle Registration Data	Spanish
Video Game Ontology	Large	68	Videogames and analytics	English

```
    [Ontology Pitfall Agent] Missing inverse relationship for playsGame.
  </vaem:rationale>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/myontology#isPlayedBy">
  <rdfs:label>is played by</rdfs:label>
  <rdfs:domain rdf:resource="http://www.semanticweb.org/myontology#Game"/>
  <rdfs:range rdf:resource="http://www.semanticweb.org/myontology#Player"/>
  <owl:inverseOf rdf:resource="http://www.semanticweb.org/myontology#playsGame"/>
  <dc:source>OOPS!: This pitfall appears when any relationship does not have an inverse
      relationship for playsGame</dc:source>
  <vaem:rationale>
    [Ontology Pitfall Agent] Created inverse property isPlayedBy for playsGame.
  </vaem:rationale>
</owl:ObjectProperty>
```

4. Evaluation

We evaluate the MASEO framework through three case studies, each driven by a CQ set of different size and domain. The goal is to analyze how the pipeline develops ontologies from CQs and how they align with the corresponding gold-standard ontologies ($Ontos_{\text{gold}}$). We first describe the experimental setup and the CQ sets used in the evaluation. We then analyze MASEO-generated ontologies ($Ontos_{\text{gen}}$) from two complementary perspectives: structural characteristics and CQ coverage derived from provenance records, and concept label matching with $Ontos_{\text{gold}}$.

4.1. Experimental Setup

4.1.1. Datasets

Table 2 summarizes the main characteristics of each CQ set used in our evaluation, all developed by knowledge engineers. The first CQ set is defined for the *Infrastructure Ontology*,[6] which focuses on infrastructure assets and their management. It includes 5 CQs, originally developed as part of the ontology design process by domain experts, and written in Spanish. The second CQ set is defined for the *Vehicle Census Ontology* (VCO).[7] It contains 28 CQs related to vehicle registration and census data, also provided in Spanish. These questions were used to guide the development of the original ontology and reflect domain-specific requirements. The third CQ set is associated with the *Video Game Ontology* (VGO),[8] which consists of 68 CQs covering core video game entities, gameplay interactions, and monetization aspects. These questions are expressed in English and originate from the CORAL corpus [47].

[6] https://github.com/telefonicasc/edint-ontologia-infraestructura
[7] https://github.com/telefonicasc/edint-ontologia-censovehiculos
[8] https://vocab.linkeddata.es/vgo/

4.1.2. Evaluation Metrics

We combine quantitative and qualitative analysis to examine the ontologies generated by MASEO. First, we analyze the structural characteristics of the generated ontologies ($Ontos_{\text{gen}}$). Quantitatively, structural metrics such as the number of classes, properties, and other ontology elements are extracted from the RDF/XML ontologies using the *owl2diagram*[9] tool and compared with those of the $Ontos_{\text{gold}}$. Qualitatively, ontology experts inspect the provenance records to explain observed structural differences and identify which CQs or repair steps led to the introduction or revision of specific concepts, properties, or axioms. Based on this analysis, we also report *CQ coverage*, defined as

$$\text{CQCoverage} = \frac{|Q_{\text{covered}}|}{|Q_{\text{input}}|}, \tag{1}$$

where Q_{input} denotes the set of input CQs and Q_{covered} denotes the subset that can be traced to at least one ontology element in the $Ontos_{\text{gen}}$ through expert inspection of the provenance records.

In the second evaluation step, we assess concept label matching between the $Ontos_{\text{gen}}$ and the $Ontos_{\text{gold}}$. Precision, recall, and F1-score are computed over concept matches obtained using three strategies, namely Exact match, lexical match, and semantic match:

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}, \quad F1 = \frac{2PR}{P + R}. \tag{2}$$

Here, TP, FP, and FN denote true positives, false positives, and false negatives. Exact match identifies concept labels that are character-for-character identical, providing a strict baseline that reflects exact terminological agreement. Lexical match is computed using *SequenceMatcher* [48], which measures character-level overlap and is well-suited for detecting near-identical labels that differ only in minor surface variations such as spacing or punctuation. Semantic similarity is computed using embedding-based cosine similarity with *embeddinggemma* [49], an open-weights model optimized for high-dimensional semantic search. This approach captures conceptual equivalence between labels that are lexically distinct but semantically related (e.g., "Authorization" and "Permit").

Together, these three strategies form a spectrum from exact to approximate matching, allowing a more comprehensive assessment of concept label alignment between the $Ontos_{\text{gen}}$ and $Ontos_{\text{gold}}$.

We also report on concept coverage under the three matching strategies:

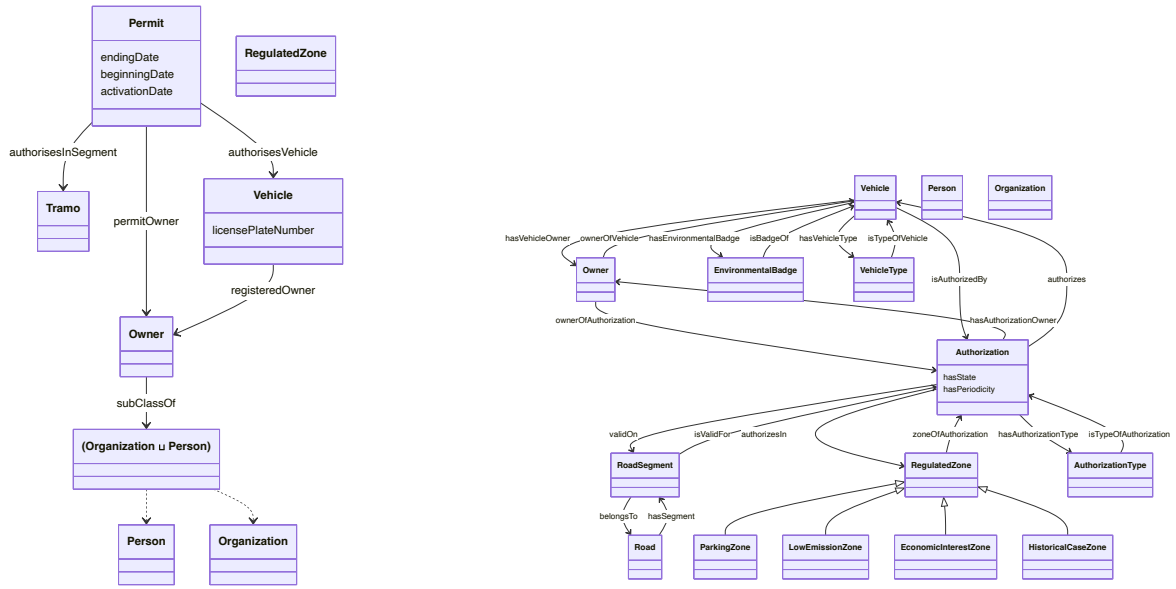
$$\text{ConceptCoverage}^{m} = \frac{|C^{m}_{\text{match}}|}{|C_{\text{gold}}|}, \quad m \in \{\text{exact}, \text{lex}, \text{sem}\}. \tag{3}$$

Here, C_{gold} is the set of classes in the $Onto_{\text{gold}}$, and C^{m}_{match} is the set of concepts matched under strategy m.

4.1.3. Reproducibility

To facilitate reproducibility, we integrate several open-source ontology engineering tools together with DeepSeek-V3.2[10] [50] for ontology generation and refinement. Syntax validation is implemented using RDFLib (v7.2.1).[11] Logical consistency checking is performed by the HermiT OWL reasoner (v1.3.8),[12] and pitfall detection is conducted via official API calls to the OOPS! [2] pitfall scanner. For the evaluation phase, ontology structural diagrams are generated using owl2diagram (v0.1.2). For lexical matching, character-level similarity is computed using *SequenceMatcher* from the `difflib` library.[13] For semantic matching, embedding-based cosine similarity is computed using *embeddinggemma*,[14] hosted locally via

[9] https://github.com/jatoledo/owl2diagram
[10] https://huggingface.co/deepseek-ai/DeepSeek-V3.2
[11] https://rdflib.readthedocs.io/en/7.1.2/
[12] http://www.hermit-reasoner.com/
[13] https://docs.python.org/3.8/library/difflib.html
[14] https://ollama.com/library/embeddinggemma



(a) Gold Standard (VCO_{gold}) (b) Generated (VCO_{gen})

Figure 2: Structural comparison of the Vehicle Census Ontology (VCO). The generated version (b) effectively identifies the core domain in (`Vehicle`, `Owner`) while exhibiting characteristic taxonomic flattening and increased property density derived from the CQs.

the *Ollama* runtime environment[15] to ensure deterministic inference. All experiments are conducted under a unified configuration across the three CQ sets to ensure comparability of results.

4.2. Results

This section presents the evaluation results of the three case studies obtained by applying MASEO. We first analyze the structural characteristics and CQ coverage derived from the provenance records.[16] This analysis is followed by an assessment of their alignment with expert-designed gold standards.

4.2.1. Structural Analysis and Competency Question Coverage

Across all CQ sets, the $Ontos_{gen}$ consistently identified the core domain entities. As illustrated in Figure 2, we use the VCO as a representative example, as its compact structure makes the comparison most interpretable; the remaining ontology diagrams are provided in Appendix B. Central entities such as `Vehicle` and `Owner` appear in both the $Ontos_{gen}$ and $Ontos_{gold}$, confirming that MASEO reliably captures the core concepts expressed in the CQs.

More broadly, Table 3 reveals consistent structural patterns across all three datasets. A recurring difference is hierarchy depth: $Ontos_{gen}$ tend to remain taxonomically shallow, strictly reflecting the concepts explicitly required by the CQs, whereas $Ontos_{gold}$ introduce richer subclass hierarchies to support broader extensibility and reusability. This contrast is most pronounced in the VGO dataset, where VGO_{gen} contains no subclass relations, while VGO_{gold} defines a 24 relation hierarchy (e.g., 14 subclasses of `Achievement`). This taxonomic divergence suggests that $Ontos_{gold}$ are designed as expansive reference models, incorporating a wider domain context that extends beyond the defined functional requirements of the CQs. The $Ontos_{gen}$ also frequently exhibit higher property density, as in the VCO case where MASEO produced 18 object properties against the 4 in gold, whereas experts tend toward more concise structures (e.g., using `ManagementRole` in $Infrastructure_{gold}$) to support broader

[15] https://ollama.com/

[16] The input CQs and the generated RDF/XML ontologies with traceable provenance records are available at https://github.com/oeg-upm/maseo/tree/main/dataset

Table 3
Comparison of structural metrics between the generated and gold-standard ontologies across three CQ-driven case studies: Infrastructure (left), Vehicle Census (middle), and Video Game (right).

	Infrastructure Ontology		Vehicle Census Ontology		Video Game Ontology	
Element	Gold	Gen	Gold	Gen	Gold	Gen
Classes	37	14	7	15	37	13
Object Properties	13	12	4	18	32	33
Datatype Properties	0	5	4	2	6	9
Subclass Relations	15	7	1	4	24	0
InverseOf Axioms	0	6	0	9	0	15
Linked CQs	5	5	28	17	68	37
CQ Coverage	100.0%		60.7%		54.4%	

extensibility. The generated ontologies also excel at capturing data attributes explicitly requested in the CQs: in the Infrastructure case, $\text{Infrastructure}_{\text{gen}}$ produced datatype properties such as `isAvailable` and `zoneName` that are absent from the $\textit{Infrastructure}_{\text{gold}}$, precisely because these were directly specified in the CQs.

Next, we examine each result in detail. The **Infrastructure Ontology**, comprising 5 CQs, yielded a focused, operational model. While $\text{Infrastructure}_{\text{gold}}$ with 37 classes is significantly larger due to its municipal scope and reuse of external vocabularies (e.g., `GeoSPARQL`, `SOSA`), $\text{Infrastructure}_{\text{gen}}$ contains 14 classes and provides a precise model directly grounded in the input requirements. Provenance records confirm 100% CQ coverage: all core classes (e.g., `ParkingLot`, `Rate`, `Permit`) are directly traceable to the input CQs, while the 6 inverse object properties were introduced exclusively by the Ontology Pitfall Agent to resolve missing inverse relationship pitfalls (P19) detected by OOPS!.

The **Vehicle Census Ontology** (28 CQs) demonstrates the ability of MASEO to handle complex relational requirements, achieving 60.7% CQ coverage. Provenance records confirm that every core entity, including specific subclasses such as `LowEmissionZone` and `RoadSegment`, was directly triggered by input CQs. The remaining divergence is due to those attributes present in VCO_{gold} (e.g., `licensePlateNumber`, `activationDate`) that were not mentioned in any of the provided CQs, and therefore fell outside the scope of MASEO. Similarly, nine inverse object properties were added by the Ontology Pitfall Agent in response to the same pitfall (P19).

The **Video Game Ontology** is the most complex case, spanning 68 CQs, with coverage decreasing to 54.4% as only 37 CQs are linked to generated ontology elements. This is not a failure of alignment but rather a consequence of semantic saturation: in larger CQ sets, multiple requirements often converge on the same underlying concept (e.g., various gameplay actions and player interactions mapped to a single `Player` or `ActionEvent` class), and the provenance records confirm that MASEO avoids redundancy by linking multiple CQs to a single, well-defined ontology element rather than generating duplicate classes. Fifteen inverse object property pairs follow the same pattern (added due to OOPS! pitfall reports).

Across all three datasets, the $\textit{Ontos}_{\text{gen}}$ closely reflect the core logic of their expert-designed gold standards, but differ in granularity and the abstraction level. A major advantage of MASEO is full auditability, as each modeling decision is documented via provenance records which allow ontology engineers edit or adapt the produced ontology.

4.2.2. Concept Label Matching

To further quantify the conceptual overlap between $\textit{Ontos}_{\text{gen}}$ and $\textit{Ontos}_{\text{gold}}$ beyond structural metrics, we next assess their alignment by matching the concept labels.

Table 4 presents the concept label matching results across the three datasets, evaluated under three distinct matching strategies. The $\textit{Ontos}_{\text{gen}}$ comprising 14, 15 and 13 classes derived from the CQs for the Infrastructure, Vehicle Census, and Video Game domains are compared against their corresponding $\textit{Ontos}_{\text{gold}}$, which contain 40, 10, and 37 classes. Concept labels are extracted by prioritizing the

Table 4
Concept label matching results across the three ontologies, comparing Onto_{gen} against $\text{Onto}_{\text{gold}}$ under three matching strategies: exact, lexical, and semantic. For each strategy, concept coverage (Cov.), precision (P), recall (R), and F1-score (F1) are reported.

	Infrastructure Ontology				Vehicle Census Ontology				Video Game Ontology			
Strategy	Cov.	P	R	F1	Cov.	P	R	F1	Cov.	P	R	F1
Generated classes			14				15				13	
Gold-standard classes			40				10				37	
Exact match	0.025	0.071	0.025	0.037	0.400	0.267	0.444	0.333	0.135	0.385	0.135	0.200
Sequence Match	0.491	0.771	0.491	0.600	0.783	0.609	0.783	0.685	0.591	0.895	0.591	0.712
Semantic match	**0.605**	**0.844**	**0.605**	**0.705**	**0.846**	**0.711**	**0.846**	**0.772**	**0.712**	**0.924**	**0.712**	**0.804**

`xml:lang="en"` attribute. In cases where no English label is available, the URI local name serves as a fallback. For the Spanish-based datasets, this mechanism retains the original Spanish identifiers for specific gold-standard classes, such as "*estacionbicicleta*" (bike station) in the Infrastructure Ontology and "*tramo*" (road segment) in the VCO.

Under Exact match, performance is consistently low across all three datasets, with F1 scores of 0.037, 0.333, and 0.200 and class coverage of 0.025, 0.400, and 0.135. Beyond the Spanish fallback identifiers, gold-standard concept labels often include externally imported identifiers (e.g., "*gsp:Feature*") that do not match the shorter, CQ-derived labels. VCO attains the highest Exact match coverage with 0.400, as several core concepts such as "*Organization*", "*Owner*", "*Person*", and "*Vehicle*, are generic enough that both MASEO and the $\textit{Ontos}_{\text{gold}}$ independently selected identical English labels.

SequenceMatcher yields intermediate results of F1 = 0.600, 0.685, and 0.712 by capturing shared substrings effectively. For instance, in the VGO, it successfully identifies the relationship between "*AchievementType*" and "*Achievement*" with similarity = 0.815, and "*VehicleType*" to "*Vehicle*" with similarity = 0.737 in the VCO. However, its purely lexical nature limits its ability to handle more significant terminological shifts.

Semantic matching achieves the highest overall performance across all three datasets, with F1 scores of 0.705, 0.772, and 0.804, and coverage reaching 0.605, 0.846, and 0.712. This superiority reflects the ability to bridge the terminological gap. For example, in the Infrastructure dataset, this metric correctly aligns "*ResidentPermit*" and "*LoadingAndUnloadingPermit*" to the gold-standard class "*Permit*" (similarity = 0.846 and 0.771 respectively). In contrast, lexical methods struggle with the string length disparity. VGO achieves the highest semantic precision in 0.924, successfully aligning generated concepts like "*In app purchase*" to the gold-standard "*In app purchase event*" (similarity = 0.878), indicating that nearly all concepts generated by MASEO are semantically valid within the target domain, even when they employ different nomenclature from the gold standard. All reported alignments were additionally reviewed by hand to confirm their quality.

Overall, the transition from exact to semantic matching consistently recovers a substantial proportion of conceptual overlap, with coverage reaching up to 0.846 in the VCO. As anticipated in our structural analysis in Section 4.2.1, the remaining coverage gap is primarily explained by the presence of gold-standard concepts representing domain knowledge beyond the scope of the input CQ set, an inconsistency already reported in prior literature [51]. This confirms that MASEO effectively maintains a "requirement-focused" modeling boundary, avoiding the inclusion of auxiliary terms found in expert-built models.

We note that this evaluation approach remains valid even when the concepts generated by MASEO diverge from those explicitly mentioned in the CQs, as semantic matching captures meaningful equivalences that lexical methods would miss.

5. Discussion

This section interprets the empirical findings and discusses the implications for multi-agent, CQ-driven ontology generation. The analysis focuses on the performance of MASEO framework, the granularity of requirements, and the challenges inherent in benchmarking against expert models.

5.1. Effectiveness of MASEO

The results demonstrate that the multi-agent architecture provides a first step towards automating the complex task of ontology development. The sequential validation stages ensure that $Ontos_{\text{gen}}$ are semantically relevant and technically sound, as evidenced by the precision of 0.92 (semantic) in the VGO case.

The strict adherence to the RDF/XML serialization was a deliberate design choice required by the OOPS! API and HermiT reasoner to achieve full pipeline automation. While this ensures a seamless end-to-end workflow, it introduces certain trade-offs in terms of output readability. Currently, MASEO only supports RDF/XML, a constraint to be addressed by integrating other RDF-based serializations like Turtle, which are easier to digest by humans.

At the moment, the scope of logical validation is defined by the OWL 2 DL profile supported by the HermiT reasoner (e.g., validating disjointness axioms). However, more complex axioms are recorded using comments, limiting consistency checking to basic structures. In future work, we plan to support formal OWL representations for complex axioms, enabling more comprehensive logical validation. In addition, we have observed that the Logical Consistency Agent may apply shallow repair strategies, for example, by removing a semantically valid disjointness axioms, particularly when CQ grounding is implicit or ambiguous. Addressing this limitation may require revisiting CQ coverage to provide the agent with stronger provenance-based grounding for its repair decisions.

As for provenance, we intend to enhance the provenance tracking mechanism from simple string annotation literals to established standards such as PROV-O [52]. This will enable the construction of structured provenance graphs, allowing the tracing of how specific CQs or agent interactions influenced the final ontology.

Finally, we acknowledge that the automated adoption of OOPS! recommendations may sometimes lead to conceptually redundant additions, such as large numbers of inverse object properties. This may be improved by considering the criticality associated with each pitfall and incorporating human preferences, allowing domain experts to evaluate the semantic necessity of suggested modifications, particularly when multiple modeling alternatives exist.

5.2. Reproducibility and Architectural Constraints

The fixed agent sequence included in this version of MASEO ensures reproducibility but introduces a single-pass execution model without iterative feedback. Once a stage is completed, MASEO does not revisit earlier outputs or re-evaluate unlinked CQs, and therefore lacks dynamic re-entry. In particular, if a core class is removed during logical consistency repair, the competency question that motivated it may become silently unlinked, with no mechanism to detect or recover this loss. While this is a deliberate trade-off to ensure computational efficiency and prevent infinite agent loops, it represents a limitation in handling the coverage of complex modeling scenarios.

In future work, we plan to explore the integration of human-in-the-loop mechanisms within an iterative refinement process, enabling the system to revise earlier modeling decisions based on human feedback and the original CQ set. This may support more robust handling of complex modeling choices and help ensure that critical competency questions are not lost during automated refinement.

Output quality also remains sensitive to model choice and prompt design. The selected open-source model (DeepSeek-V3.2) requires significant computational infrastructure to run, and minor variations in the framing of pitfall reports may lead to different structural adjustments, underscoring the challenge of achieving fully deterministic automated modeling [17, 53].

5.3. Influence of CQ Set Scale and Quality

Our experiments reveal a trade-off between CQ set scale and ontology richness. The Infrastructure CQ set, comprising 5 CQs, achieved 100% CQ coverage but showed the lowest structural similarity to $Onto_{\text{gold}}$, reflecting a sparse but requirement-faithful model where every CQ is explicitly mapped to a concept. In contrast, the VGO CQ set, spanning 68 CQs, yielded a richer ontology but lower CQ coverage of 54.4%, as semantic overlap across requirements allows a small number of core elements to resolve multiple CQs, resulting in a more condensed conceptual model.

CQ quality also affects modeling accuracy. CQs containing examples led MASEO to interpret instance-level data as named subclasses rather than individuals. This schema–instance conflation is a systematic limitation rather than an isolated edge case; without a principled mechanism to separate TBox concepts from ABox instances during input processing, the generation agent struggles to reliably distinguish conceptual structure from exemplary data. In future work, we plan to investigate the impact of incorporating a semantic interpretation step that structures CQ inputs prior to generation.

5.4. Rethinking Benchmarks for CQ-Driven Ontology Generation

The comparison between $Onto_{\text{gen}}$ and $Onto_{\text{gold}}$ reveals a fundamental divergence in design philosophy. As noted in Section 4.2.1, $Onto_{\text{gold}}$ often function as expansive reference models that incorporate external vocabularies such as SOSA and GeoSPARQL, which extend well beyond the scope of the input CQs. The observed gaps in semantic coverage therefore, reflect the relative over-completeness of expert models with respect to the input requirements, rather than a deficiency in the MASEO generation process. This discrepancy suggests that when $Onto_{\text{gold}}$ encodes knowledge not present in the input CQs, absolute structural alignment becomes a misleading metric for pipeline quality. This motivates the need for dedicated benchmarks where gold-standard references are derived strictly from the same CQ sets. Such a framework would enable a fair assessment based on requirement fit, distinguishing between a failure to capture stated requirements and the intentional omission of out-of-scope domain knowledge.

Beyond structural metrics, future work should also explore more advanced evaluation strategies that go beyond label matching. In particular, we aim to consider not only concept labels but also the relationships and properties between them, enabling a more faithful assessment of the semantic validity of generated ontologies.

6. Conclusion

This paper presents MASEO, a fully automated end-to-end multi-agent framework for ontology generation from Competency Questions. By decomposing the ontology engineering workflow into four specialized stages, MASEO generates OWL ontologies directly from CQs and incrementally refines them through syntax repair, logical consistency verification, and pitfall resolution. A key contribution of MASEO is a provenance tracking mechanism, which links each ontology term and axiom to the specific CQ or validation feedback that motivated its introduction or revision. An evaluation across three case studies shows that the proposed framework provides a first step towards operationalizing input requirements. The results further indicate that CQ coverage and structural alignment with expert-designed ontologies represent complementary dimensions of evaluation. Overall, MASEO improves the transparency, traceability, and practical usefulness of automated ontology generation.

More broadly, MASEO is designed to support knowledge engineers. Instead of producing final ontologies, it provides a transparent and traceable starting point that can be refined and validated by human experts after each iteration. The provenance tracking mechanism is central to this approach, as it makes each automated decision visible and traceable, allowing engineers to understand, challenge, and revise modeling choices. In this way, generative models act as assistants rather than decision-makers in the construction of shared conceptualizations.

As for future work, we plan to extend MASEO to better support additional stages of the ontology development lifecycle [1] (e.g., ontology reuse). Given the flexibility of our framework in defining new tasks and agents, we intend to explore alternative agentic configurations that enable CQ coverage validation and human-in-the-loop refinement after each iteration (i.e., after going through the full pipeline). We will also investigate methods for generating complex OWL axioms and for separating schema-level concepts from instance-level data through CQ pre-processing. To further enhance transparency, we will align our provenance graphs with the PROV-O standard, producing structured, queryable traces of how each axiom and term was derived from specific competency questions or validation feedback.

Acknowledgments

This work was supported by the grant "SOEL: Supporting Ontology Engineering with Large Language Models" PID2023-152703NA-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF/UE". The authors would also like to thank the EDINT (Espacios de Datos para las Infraestructuras Urbanas Inteligentes) ontology development team for sharing the project resources for evaluation purposes.

Declaration on Generative AI

All the ideas presented in this manuscript are original from the authors. During the preparation of this work, the authors used Claude (Anthropic) to improve the clarity, grammar, and readability of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] M. Poveda-Villalón, A. Fernández-Izquierdo, M. Fernández-López, R. García-Castro, LOT: An industrial oriented ontology engineering framework, Engineering Applications of Artificial Intelligence 111 (2022) 104755. doi:10.1016/j.engappai.2022.104755.

[2] M. C. Suárez-Figueroa, A. Gómez-Pérez, Towards a glossary of activities in the ontology engineering field, in: Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08), European Language Resources Association (ELRA), Marrakech, Morocco, 2008. URL: https://aclanthology.org/L08-1127/.

[3] N. F. Noy, D. L. McGuinness, et al., Ontology development 101: A guide to creating your first ontology, 2001. URL: http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html.

[4] M.-J. Antia, C. M. Keet, Automating the generation of competency questions for ontologies with AgOCQs, in: Iberoamerican Knowledge Graphs and Semantic Web Conference, Springer, 2023, pp. 213–227. doi:10.1007/978-3-031-47745-4_16.

[5] V. K. Kommineni, B. König-Ries, S. Samuel, From human experts to machines: An LLM supported approach to ontology and knowledge graph construction, arXiv preprint arXiv:2403.08345 (2024). doi:10.48550/arXiv.2403.08345.

[6] R. Alharbi, V. Tamma, F. Grasso, T. Payne, An experiment in retrofitting competency questions for existing ontologies, in: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, 2024, pp. 1650–1658. doi:10.1145/3605098.3636053.

[7] M. Llugiqi, F. J. Ekaputra, M. Sabou, From experts to LLMs: Evaluating the quality of automatically generated ontologies, in: Proceedings of the 2nd Workshop on Evaluation of Language Models in Knowledge Engineering (ELMKE) co-located with ESWC 2025, volume 3977 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. URL: https://ceur-ws.org/Vol-3977/elmke-1.pdf.

[8] R. R. Chowdhury, T. Goto, K. Tsuchida, T. Kirishima, A. Bandi, An automated framework of ontology generation for abstract concepts using LLMs, in: International Conference on Computers and Their Applications, Springer, 2025, pp. 170–180. doi:10.1007/978-3-031-92178-0_14.

[9] S. Bischof, E. Filtz, J. X. Parreira, S. Steyskal, LLM-based guided generation of ontology term definitions, in: European Semantic Web Conference, Springer, 2024, pp. 133–137. doi:10.1007/978-3-031-78952-6_13.

[10] S. Toro, A. V. Anagnostopoulos, S. M. Bello, K. Blumberg, R. Cameron, L. Carmody, A. D. Diehl, D. M. Dooley, W. D. Duncan, P. Fey, et al., Dynamic retrieval augmented generation of ontologies using artificial intelligence (DRAGON-AI), Journal of Biomedical Semantics 15 (2024) 19. doi:10.1186/s13326-024-00320-3.

[11] S. Tsaneva, S. Vasic, M. Sabou, LLM-driven ontology evaluation: Verifying ontology restrictions with ChatGPT, in: Proceedings of the Workshop on Data Quality meets Machine Learning and Knowledge Graphs (DQMLKG) co-located with ESWC 2024, volume 3747 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3747/dqmlkg_paper3.pdf.

[12] A. S. Lippolis, M. J. Saeedizade, R. Keskisärkkä, A. Gangemi, E. Blomqvist, A. G. Nuzzolese, Large language models assisting ontology evaluation, in: International Semantic Web Conference, Springer, 2025, pp. 502–520. doi:10.1007/978-3-032-09527-5_27.

[13] B. Zhang, V. A. Carriero, K. Schreiberhuber, S. Tsaneva, L. S. González, J. Kim, J. de Berardinis, OntoChat: A framework for conversational ontology engineering using language models, in: The Semantic Web: ESWC 2024 Satellite Events, Springer Nature Switzerland, Cham, 2025, pp. 102–121. doi:10.1007/978-3-031-78952-6_10.

[14] M. T. Pazienza, A. Stellato, Semi-Automatic Ontology Development: Processes and Resources, IGI Global, 2012. doi:10.4018/978-1-4666-0188-8.

[15] D. Garijo, O. Corcho, M. Poveda-Villalón, FOOPS!: An ontology pitfall scanner for the FAIR principles, in: Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks, volume 2980 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021. URL: https://ceur-ws.org/Vol-2980/paper321.pdf.

[16] S. Tsaneva, K. Käsznar, M. Sabou, Human-centric ontology evaluation: Process and tool support, in: O. Corcho, L. Hollink, O. Kutz, N. Troquard, F. J. Ekaputra (Eds.), Knowledge Engineering and Knowledge Management, Springer International Publishing, Cham, 2022, pp. 182–197.

[17] Z. Liu, Y. Zhou, Y. Zhu, J. Lian, C. Li, Z. Dou, D. Lian, J.-Y. Nie, Information retrieval meets large language models, in: Companion Proceedings of the ACM Web Conference 2024, 2024, pp. 1586–1589. doi:10.1145/3589335.3641299.

[18] X. Wang, Q. Yang, Y. Qiu, J. Liang, Q. He, Z. Gu, Y. Xiao, W. Wang, KnowledGPT: Enhancing large language models with retrieval and storage access on knowledge bases, arXiv preprint arXiv:2308.11761 (2023). doi:10.48550/arXiv.2308.11761.

[19] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Advances in Neural Information Processing Systems 35 (2022) 24824–24837. doi:10.48550/arXiv.2201.11903.

[20] S. Diao, P. Wang, Y. Lin, R. Pan, X. Liu, T. Zhang, Active prompting with chain-of-thought for large language models, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 1330–1350. doi:10.18653/v1/2024.acl-long.73.

[21] X. Pan, J. v. Ossenbruggen, V. de Boer, Z. Huang, A RAG approach for generating competency questions in ontology engineering, in: Research Conference on Metadata and Semantics Research, Springer, 2024, pp. 70–81. doi:10.1007/978-3-031-81974-2_6.

[22] Z. Mahlaza, C. M. Keet, N. Chahinian, B. Haydar, On the feasibility of LLM-based automated generation and filtering of competency questions for ontologies, in: Proceedings of the 5th Conference on Language, Data and Knowledge, 2025, pp. 136–146. URL: https://aclanthology.org/2025.ldk-1.15/.

[23] Y. Rebboud, P. Lisena, L. Tailhardat, R. Troncy, Benchmarking LLM-based ontology conceptualization: A proposal, in: Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024), volume 3953 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3953/365.pdf.

[24] M. L. Coutinho, Leveraging LLMs in text-based ontology-driven conceptual modeling, 2024. URL: https://www.utwente.nl/en/eemcs/fois2024/resources/papers/

coutinho-leveraging-llms-in-text-based-ontology-driven-conceptual-modeling.pdf.
[25] G. Kholmska, K. Kenda, J. Rozanec, Enhancing ontology engineering with LLMs: From search to active learning extensions, Proceedings of Data Mining and Data Warehauses – Sikdd 2024 (2024). URL: https://doi.org/10.70314/is.2024.sikdd.28.
[26] A. Pisu, L. Pompianu, A. Salatino, F. Osborne, D. Riboni, E. Motta, D. R. Recupero, Leveraging language models for generating ontologies of research topics, in: Proceedings of the 3rd International Workshop on Knowledge Graph Generation from Text (Text2KG 2024) co-located with ESWC 2024, volume 3747 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3747/text2kg_paper6.pdf.
[27] N. Fathallah, A. Das, S. D. Giorgis, A. Poltronieri, P. Haase, L. Kovriguina, NeOn-GPT: a large language model-powered pipeline for ontology learning, in: European Semantic Web Conference, Springer, 2024, pp. 36–50. doi:10.1007/978-3-031-78952-6_4.
[28] M. Gruninger, Methodology for the design and evaluation of ontologies, in: Proc. IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, 1995. URL: http://citeseer.ist.psu.edu/grninger95methodology.html.
[29] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: an OWL 2 reasoner, Journal of Automated Reasoning 53 (2014) 245–269. doi:10.1007/s10817-014-9305-1.
[30] M. Poveda-Villalón, A. Gómez-Pérez, M. C. Suárez-Figueroa, OOPS! (Ontology Pitfall Scanner!): An on-line tool for ontology evaluation, International Journal on Semantic Web and Information Systems (IJSWIS) 10 (2014) 7–34. doi:10.4018/ijswis.2014040102.
[31] J. Li, Z. Wang, D. Garijo, M. Poveda-Villalón, oeg-upm/maseo: V1.1 — MASEO: Multi Agent System for Explainable Ontology generation, 2026. URL: https://doi.org/10.5281/zenodo.19052003. doi:10.5281/zenodo.19052003.
[32] A. Fernández-Izquierdo, M. Poveda-Villalón, R. García-Castro, CORAL: A corpus of ontological requirements annotated with lexico-syntactic patterns, Springer-Verlag, Berlin, Heidelberg, 2019. doi:10.1007/978-3-030-21348-0_29.
[33] D. Garijo, M. Poveda-Villalón, E. Amador-Dominguez, Z. Wang, R. García-Castro, O. Corcho, LLMs for ontology engineering: A landscape of tasks and benchmarking challenges, in: Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024), volume 3953 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3953/364.pdf.
[34] O. Perera, J. Liu, Exploring large language models for ontology learning, Issues in Information Systems 25 (2024) 299–310. doi:10.48009/4_iis_2024_124.
[35] Y. Tang, A. A. B. Da Costa, X. Zhang, I. Patrick, S. Khastgir, P. Jennings, Domain knowledge distillation from large language model: An empirical study in the autonomous driving domain, in: 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2023, pp. 3893–3900. doi:10.1109/ITSC57777.2023.10422308.
[36] H. Jadhav, T. Heger, B. König-Ries, A. Algergawy, Ontology evolution in invasion biology using large language models: A hybrid approach, in: Proceedings of the 4th International Workshop on LLM-Integrated Knowledge Graph Generation from Text (TEXT2KG 2025), volume 4020 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. URL: https://ceur-ws.org/Vol-4020/Paper_ID_13.pdf.
[37] D. Doumanas, A. Soularidis, K. Kotis, G. Vouros, Integrating LLMs in the engineering of a SAR ontology, in: IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer, 2024, pp. 360–374. doi:10.1007/978-3-031-63223-5_27.
[38] J. H. Caufield, H. Hegde, V. Emonet, N. L. Harris, M. P. Joachimiak, N. Matentzoglu, H. Kim, S. Moxon, J. T. Reese, M. A. Haendel, et al., Structured prompt interrogation and recursive extraction of semantics (SPIRES): A method for populating knowledge bases using zero-shot learning, Bioinformatics 40 (2024) btae104. doi:10.1093/bioinformatics/btae104.
[39] N. Tufek, A. Saissre, A. Hanbury, Validating semantic artifacts with large language models, in: Proceedings of the 21th European Semantic Web Conference (ESWC), Krete, Greece, 2024, pp. 24–30. doi:10.1007/978-3-031-78952-6_9.
[40] M. Val-Calvo, M. E. Aranguren, J. Mulero-Hernández, G. Almagro-Hernández, P. Deshmukh,

J. A. Bernabé-Díaz, P. Espinoza-Arias, J. L. Sánchez-Fernández, J. Mueller, J. T. Fernández-Breis, OntoGenix: Leveraging large language models for enhanced ontology engineering from datasets, Information Processing & Management 62 (2025) 104042. doi:10.1016/j.ipm.2024.104042.

[41] A. S. Lippolis, M. Ceriani, S. Zuppiroli, A. G. Nuzzolese, Ontogenia: Ontology generation with metacognitive prompting in large language models, in: European Semantic Web Conference, Springer, 2024, pp. 259–265. doi:10.1007/978-3-031-78952-6_38.

[42] Z. Ouédraogo, L. S. Tapsoba, A. Sabane, R. Kafando, A. K. Kabore, T. F. Bissyande, Text-to-OWL: Automated ontology construction for tuberculosis treatment recommendation using generative AI, in: T. Koné, A. Sere, K. F. Kouamé (Eds.), Towards New e-Infrastructure and e-Services for Developing Countries, Springer Nature Switzerland, Cham, 2025, pp. 281–294. doi:10.1007/978-3-032-01910-3_18.

[43] P. Haase, D. M. Herzig, A. Kozlov, A. Nikolov, J. Trame, metaphactory: A platform for knowledge graph management, Semantic Web 10 (2019) 1109–1125. doi:10.3233/SW-190360.

[44] T. I. Ivanova, Large language models and ontology learning, in: 2025 International Conference on Information Technologies (InfoTech), 2025, pp. 1–4. doi:10.1109/InfoTech67177.2025.11175968.

[45] S. Chari, O. Seneviratne, M. Ghalwash, S. Shirai, D. M. Gruen, P. Meyer, P. Chakraborty, D. L. McGuinness, Explanation ontology: A general-purpose, semantic representation for supporting user-centered explanations, Semantic Web 15 (2024) 959–989. doi:10.3233/SW-233282.

[46] A. Tang, Y. Jin, J. Han, A rationale-based architecture model for design traceability and reasoning, Journal of Systems and Software 80 (2007) 918–934. doi:10.1016/j.jss.2006.08.040.

[47] A. Fernández-Izquierdo, M. Poveda-Villalón, R. García-Castro, CORAL: a corpus of ontological requirements annotated with lexico-syntactic patterns, in: European Semantic Web Conference, Springer, 2019, pp. 443–458. doi:10.1007/978-3-030-21348-0_29.

[48] J. W. Ratcliff, D. E. Metzener, et al., Pattern matching: The gestalt approach, Dr. Dobb's Journal 13 (1988) 46.

[49] H. S. Vera, S. Dua, B. Zhang, D. Salz, R. Mullins, S. R. Panyam, S. Smoot, I. Naim, J. Zou, F. Chen, et al., EmbeddingGemma: Powerful and lightweight text representations, arXiv preprint arXiv:2509.20354 (2025). doi:10.48550/arXiv.2509.20354.

[50] A. Liu, A. Mei, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong, et al., DeepSeek-V3.2: Pushing the frontier of open large language models, arXiv preprint arXiv:2512.02556 (2025). doi:10.48550/arXiv.2512.02556.

[51] A. Fernández-Izquierdo, M. Poveda-Villalón, R. García-Castro, Analysing ontological requirements: A journey from requirements to code and back, in: XIX Conference of the Spanish Association for Artificial Intelligence (CAEPIA 20/21), Málaga, Spain, 2021.

[52] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, J. Zhao, PROV-O: The PROV ontology (2013). URL: https://www.w3.org/TR/prov-o/.

[53] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D. C. Schmidt, A prompt pattern catalog to enhance prompt engineering with ChatGPT, arXiv preprint arXiv:2302.11382 (2023). doi:10.48550/arXiv.2302.11382.

A. Agent Configuration and System Prompts

This section provides the detailed configurations for the specialized agents within the MASEO pipeline. Table 5 summarizes the roles and inputs for each stage. The specific system instructions used to guide the LLM-based agents are detailed below to ensure experimental reproducibility.

Ontology Generation Agent Instruction:

"You are an expert knowledge engineer specializing in domain modeling. Based on the provided CQs, construct a formal ontology in OWL format using RDF/XML syntax. Identify all relevant concepts, properties, domains, and ranges necessitated by the requirements, ensuring the output is a valid XML-structured ontology."

Table 5
Summary of the specialized agents, defined roles, and input sources for each stage of the pipeline.

Pipeline Stage	Assigned Role	Primary Input Source
Ontology Generation	Expert Knowledge Engineer	Competency Questions
Syntax Repair	Technical Knowledge Engineer	OWL Ontology and RDFLib Error Logs
Logical Consistency	Expert Ontology Engineer	OWL Ontology and HermiT Reports
Pitfall Resolution	Expert Ontology Engineer	OWL Ontology and OOPS! Evaluation

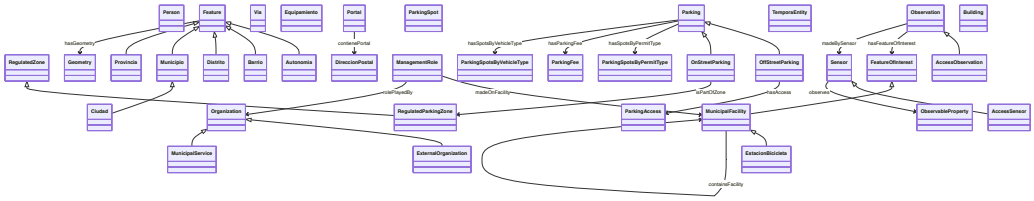
Syntax Repair Agent Instruction:
"You are a technical knowledge engineer. Your task is to identify and resolve syntax errors in the provided RDF/XML source code to ensure the file is parsable by standard RDF libraries."

Logical Consistency Agent Instruction:
"You are an expert ontology engineer. You are provided with a debugging report from the HermiT reasoner for a specific ontology. Your objective is to analyze the identified inconsistencies or subsumption errors, refine the ontology source code accordingly, and output a logically sound version."

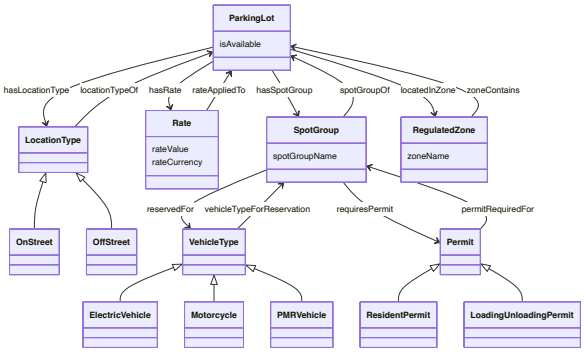
Ontology Pitfall Agent Instruction:
"You are an expert ontology engineer. You are provided with a diagnostic report from OOPS! (Ontology Pitfall Scanner). Your task is to evaluate the identified modeling pitfalls, implement necessary architectural repairs when appropriate, and generate an optimized version of the ontology."

B. Ontology Structural Diagrams

To complement the quantitative structural analysis presented in Section 4.2, this section provides visual comparisons between the expert-built gold standards and the ontologies generated by MASEO. These diagrams illustrate the hierarchical depth and conceptual coverage discussed in the main text.

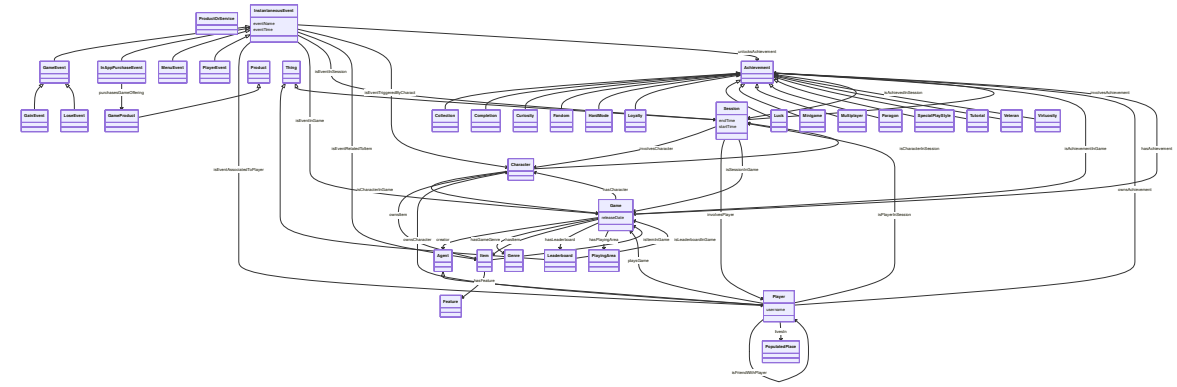


(a) Gold Standard (Infrastructure$_{\text{gold}}$)

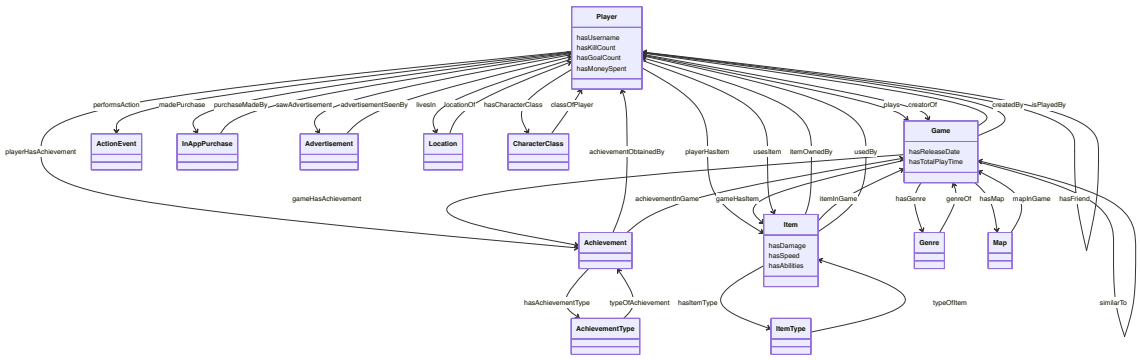


(b) Generated (Infrastructure$_{\text{gen}}$)

Figure 3: Structural comparison of the Infrastructure Ontology. Infrastructure$_{\text{gold}}$ incorporates a broader municipal scope with external vocabulary reuse, while Infrastructure$_{\text{gen}}$ provides a focused model grounded in the five input CQs.



(a) Gold Standard (VGO_{gold})



(b) Generated (VGO_{gen})

Figure 4: Structural comparison of the Video Game Ontology (VGO). VGO_{gold} defines a rich hierarchy with 24 relational subclasses, whereas VGO_{gen} remains taxonomically flat with higher property density derived from the 68 input CQs.