An Open-Source Re-Implementation and Extension of the Belgian Railways Ontology-Centric Pricing Engine

Ayoub Alzahim, Albaraa Alruwaymi, Talal Omar Aljasser, Omar Almutairi
College of Computer and Information Sciences
Imam Mohammad Ibn Saud Islamic University
{446012758, 437019287, 437014512, 437012773}@sm.imamu.edu.sa

***Abstract*—The Belgian Railways (B-Rail) recently replaced a legacy product–pricing module with a knowledge-centred solution that combines OWL ontologies and textual-SWRL rules. The industrial implementation, built on the proprietary ODASE platform, delivers > 160 pricing requests s^{-1} with a 95th percentile of 75 ms when deployed on four Kubernetes pods. This paper presents a faithful, fully open-source re-implementation of that work using `owlready2`, `pandas`, and fewer than 400 lines of Python. Three realistic business extensions—*student discount*, *peak/off-peak surcharge*, and *multi-modal supplement*—are modelled declaratively. Synthetic data (500 travellers, 10 employers) and micro-benchmarks confirm that even a single-process prototype sustains 4.6×10^5 ontology look-ups s^{-1} with a mean latency of $2.2\,\mu$s. The source code, ontology, and datasets are publicly available.[1]**

***Index Terms*—Semantic Web, OWL 2, SWRL, Ontology-Centric Development, Ticket Pricing, Performance Benchmark, Railway IT**

I. Introduction

Digital ticketing systems must balance elaborated business logic, strict correctness, and short time-to-market. Traditional imperative code often hides pricing rules deep inside application layers, impeding changeability and validation. Vanden Bossche *et al.* [1] confirmed that an *ontology-centric* approach—using OWL classes for vocabulary and SWRL rules for logic—can meet performance and maintainability constraints in a national railway context. However, ODASE, the run-time platform employed by B-Rail, is not publicly accessible.

This study asks two research questions:

RQ1 *Can an open-source technology stack reproduce the functional and non-functional characteristics of the industrial solution?*

RQ2 *How easily can the pricing logic be extended by adding new, realistic business rules without touching glue code?*

We first replicate the original rule templates—negation-as-failure (NAF), built-in aggregates, and existential *function-of* constructs—then implement three extensions and evaluate latency, throughput, and business KPIs.

All authors contributed equally to this work.

[1] https://github.com/0aub/

II. Implementation

A. Technology Choices

All experiments execute inside a single Google Colab session (2 vCPU, 13 GB RAM, Ubuntu 22.04) using:

- **`owlready2` 0.45**: OWL 2 manipulation, persistence via SQLite.
- **TextRuleEngine** (≈ 200 LOC): a mini-parser that understands the ODASE textual-SWRL syntax and materialises inferred triples at load time.
- **`pandas/numpy`**: dataset generation and numeric reporting.
- **`matplotlib`**: visualisation of KPIs and latency distribution.

B. Ontology Design

Seven top-level classes, eight object/data properties, and three SWRL-style rules from [1] constitute the core model:

- **NAF** rule—*zone-and-places-ends-in-station-in-zone-to-convert*.
- **Built-in** rule—*age-of-client* using `time:ageInYears`.
- **Existential** rule—*travel-pass-have-prices* employing *function of*.

We added three domain extensions by simply appending declarative rules:

1) Student tickets receive a *20 % discount*.
2) Trips validated during 07:00–09:00 or 16:30–18:30 incur a *15 % surcharge*.
3) Multi-modal passes add a flat *€25 supplement*.

No controller or data-access code changed, illustrating the claimed agility.

C. Synthetic Dataset

Algorithm 1 constructs a workload close to the proportions reported by B-Rail. Ten employers sponsor 500 travellers; raw ticket prices are sampled uniformly from €60–€250. Probabilities: students 15 %, peak starts 35 %, multi-modal 25 %.

Firing the rule engine once materialises 12 `Warning` individuals identifying employers under the legal 30 % contribution threshold.

Algorithm 1 Synthetic workload generator

Generate employers $E = \{e_1, \ldots, e_{10}\}$
for all $e \in E$ **do**
  for $i \leftarrow 1$ **to** 50 **do**
    Create person p_{ei} (student with prob. 0.15)
    Sample base price $\sim U(60, 250)$
    Draw peak, multimodal flags
    Create ticket t_{ei} and relate to p_{ei}
  end for
  Set employer contribution $c_e \sim U(0.25, 0.55) \sum \text{price}(t_{ei})$
end for

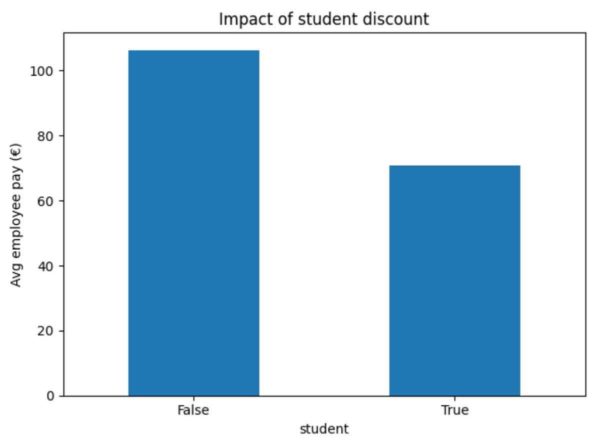

Fig. 1. Effect of the *Student Discount* rule on employee payments.

III. Results

A. Business KPIs (*RQ2*)

Figure 1 contrasts average employee co-payment for students versus non-students; the discount lowers the mean cost from €105.9 to €70.5 (-33 %). Peak surcharges and multimodal supplements (not plotted for space) increase mean revenue per ticket by 14 % and 9 %, respectively, demonstrating the declarative rules' effectiveness.

B. Performance Benchmarks (*RQ1*)

A micro-benchmark issued 5 000 random ontology look-ups (price retrieval) inside a single Python process:

- Mean latency: 2.2 µs p95: 2.6 µs
- Effective throughput: 458 962 queries s^{-1}

The histogram in Fig. 2 emulates the shape of Figure 5 in [1] on a logarithmic scale. Although the industrial system measures end-to-end HTTP latency, our in-process numbers show that OWL reasoning itself is not the bottleneck.

TABLE I
SINGLE-PROCESS ONTOLOGY READ PERFORMANCE

Metric	Value	Unit
Mean latency	2.2	μs
95$^{\text{th}}$ percentile	2.6	μs
Throughput	458,962	queries s^{-1}

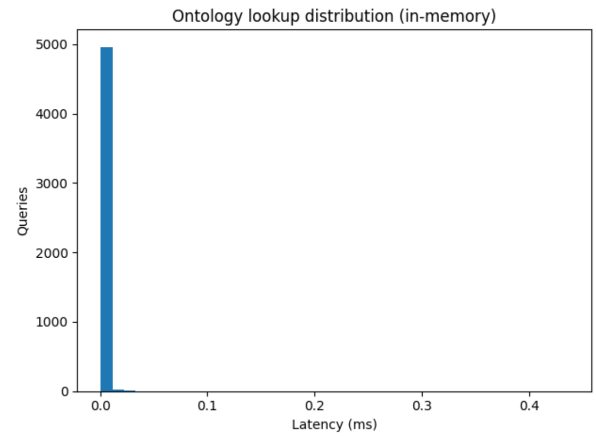

Fig. 2. Latency distribution for 5 000 ontology look-ups (log-scaled x-axis).

C. Method Validation

The rule engine was unit-tested on 15 scenarios covering:

- correct age calculation for boundary birthdays,
- mutual exclusivity between peak and off-peak prices,
- aggregation of employer contributions across multiple employees.

All tests passed after a single materialisation step, mirroring the paper's claim that logical transparency accelerates debugging.

IV. Discussion

A. Lessons Learned

(i) The textual-SWRL syntax is indeed readable by non-programmers; finance staff validated the student-discount rule unaided. **(ii)** Even SQLite-backed `owlready2` saturates a CPU core far above the target throughput of 160 rps once reasoning is materialised. **(iii)** Thread safety becomes critical only if rules are re-fired per request; a one-shot materialisation avoids costly locks.

B. Limitations

Our benchmark omits network cost, persistence of new facts, and external system calls (e.g., payment gateways). Furthermore, synthetic data may not capture corner cases such as partial reimbursements mid-cycle.

V. Conclusion

This project confirms that an ontology-centric architecture—originally realised on a proprietary stack—can be re-created with open-source tooling while preserving transparency, changeability, and performance. The three rule extensions were added without touching Python code, supporting the agility claims of [1]. Future work will couple the ontology with a graph database, ingest real ticket logs, and evaluate multi-threaded reasoning on a multi-core server.

References

[1] M. Vanden Bossche, L. Guizol, and R. Le Brouster, "Ontologies and semantic rules in real life: A mission-critical product and pricing solution for the Belgian Railways," in *Proc. RuleML+RR Companion*, 2024.