EasyChair Preprint

№ 15236

Ontologies and Semantic Rules in Real Life

Michel Vanden Bossche-Marquette, Léa Guizol and
Rémi Le Brouster

October 18, 2024

Ontologies and Semantic Rules in Real Life

A Mission-Critical Product and Pricing Solution for the Belgian Railways

Michel Vanden Bossche-Marquette*,†, Léa Guizol† and Rémi Le Brouster†

ODASE SRL, Square de Meeûs 38-40, 1000 Brussels, Belgium

Abstract

Software is often delivered late, over budget, prone to errors, and at risk of serious negative outcomes. This article presents the successful delivery of a digital solution for the Belgian Railways, from requirements specification to roll-out in the cloud. This was achieved using an ontology-centric software development and execution platform (ODASE) combining OWL-RDF ontologies and Semantic Rules based on SWRL with extensions. The semantic model allows for a complete description and execution of business requirements, and programming is only required for the technical scaffolding necessary for a fully working solution. This mission-critical application has been in production for more than a year and is successful on all fronts: correctness, transparency, changeability, time-to-market, cost effectiveness and performance.

Keywords

Ontologies, SWRL, ontology-centric development, software engineering, transport

1. Business Case

Due to changing market conditions in the post-pandemic era, the Belgian Railway Operator (BRail) wanted to introduce a new flexible season ticket [1] for hybrid working train travelers. Since the legacy product and pricing system was out-of-date, and as Sales and Marketing were looking for maximum agility and minimum time-to-market, the decision was made to consider a bespoke solution based on ontology-centric development.

2. Challenges

The business side of the company required 1) agility to quickly adapt the product and prices to new market conditions, 2) a short time-to-market for the first release and following iterations, 3) supporting combined transport (customers buying, in one transaction, BRail products and those from other Public Transport Operators), and 4) cost effectiveness in light of unknown demand.

The IT side of the company required 1) integration within a complex landscape of existing systems and applications, 2) compliance with OpenAPI [2] and the internal DevOps methodology and infrastructure, 3) deployment in the cloud using Kubernetes and containerized applications, and 4) no performance penalty for maximum users' satisfaction.

3. Rule-Based Solution

3.1. Introduction

The business logic of the new flexible season ticket is entirely expressed by a set of OWL ontologies and SWRL [3] logical rules (see section 3.2).

RuleML+RR'24: Companion Proceedings of the 8th International Joint Conference on Rules and Reasoning, September 16—22, 2024, Bucharest, Romania

* Corresponding author

† These authors contributed equally

michel.vandenbossche@odase.io; lea.guizol@odase.io; remi.le.brouster@odase.io

These ontologies and rules define products, generate quotes, and calculate the prices of the flexible season ticket. In a Business-to-Business context, the customer is an employer or a governmental organization and contributes to the payment of the ticket. The system distributes the total price between the employers and the employee, applying, when applicable, a legal minimum contribution from the employers. It also calculates the reimbursement during the life-cycle of the season ticket.

The company Odase was chosen to develop the ontologies in interaction with subject matter experts, to provide the OpenAPIs needed by IT for integration with other systems and to deliver a run-time compatible with BRail's cloud containerized environment, consistent with their DevOps procedures and infrastructure.

A key technical component of the project is the ODASE platform for ontology-centric developments. It is developed using the Mercury logic/functional language [4] to maximize quality and performance, and minimize the impedance mismatch between the logic world of ontologies and rules, and the imperative world of mainstream programming.

3.2. Extended SWRL

When we want to implement operational ontologies that clearly and completely separate the definition of the business problem from the technical implementation, the ontology must be able to specify 100% of the business logic. We cannot therefore limit ourselves to the structural part of the ontology (OWL, RDF): the business logic must be tightly associated with the ontology. This is the role of SWRL designed to enable declarative assertions using OWL concepts and properties. Without SWRL, part of the business logic inevitably requires programming – using mainstream languages, libraries and frameworks – outside of the ontology, which would no longer guarantee a complete separation between the declarative definition of the problem and the imperative implementation of the solution. We then relapse into known problems associated with a partial modeling of the problem, typical of Computer-Aided Software Engineering, Model Driven Engineering and Semantic Web Technologies limited to OWL, RDF and SPARQL: parts of the business logic are programmed and no longer understandable by the business, nor testable and explainable (see section 3.3).

Our experience led us to evolve SWRL in two directions: 1) to give SWRL a textual syntax that is understandable by business experts, and 2) expressiveness extensions (aggregates, NAF and existentials). Figure 1 is an example of *NAF* (Negation as Failure) and Figure 2 an example of *function of* (existential). Both examples show the textual syntax.

```
82 rule zone-and-places-ends-in-station-in-zone-to-convert
83 if ?product is a p:Product
84     and ?product p:endsInAskedPlace ?station
85     and ?station p:isIncludedInZone ?zone
86     and NAF (?product p:endsInZone false)
87 then ?product p:endsIn ?zone
88 .
```

Figure 1: Example of Negation as Failure

```
rule travel-pass-have-prices
if ?pass is a p:TravelAbo
    and ?pass is a b:ProductWithCalculablePrice
    and ?tPrice function of [b:CalculatedPrice, ?pass]
then
    ?tPrice is a b:TravelAboPrice
    and ?tPrice is a b:CalculatedPrice
    and ?pass b:hasCalculatedPrice ?tPrice
    and ?pass b:hasTravelAboPrice ?tPrice
.
```

Figure 2: Example of Function Of

The ontology, stricto sensu (OWL, RDF), stays generic, and can be used for other tasks.

3.3. Testing and Explanation

Since the ontology is an executable model, it can be tested before the first line of code is written. The results of these tests are explainable thanks to the logic foundations of the Semantic Web Language and the ODASE declarative debugger, with an example shown in Figure 3.

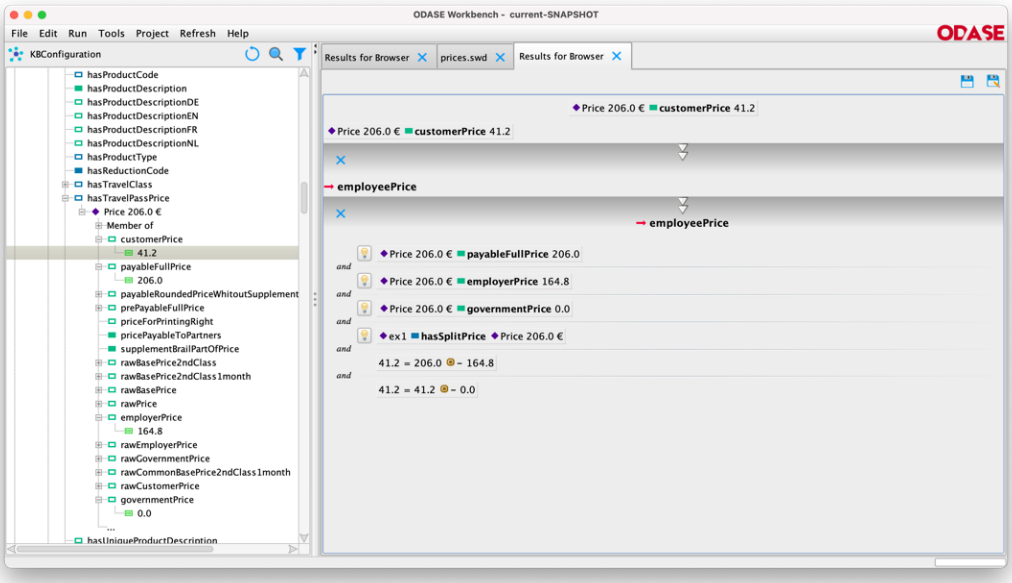


Figure 3: ODASE declarative debugger (or *explainer*)

As a result, fixing conceptual bugs at a very early stage is easy. This is not the case with conventional software development, with the side effects associated with classical programming compounding the problem. Also, explanations are of primary importance: they help subject matter experts deepen their understanding of the problem, contributing to a specification that is correct, complete and unambiguous, using a fast iterative process. This has a major positive impact on cost, speed of development, quality and agility.

3.4. Automatic Code Generation and Low Code

Unlike other requirements specification, the ontology is not used as a reference for writing the application by hand. The ODASE platform includes a tool that automatically generates a Java API from an ontology. It creates a Java class for each OWL concept and Java getters and

setters for each OWL property. OWL subclass axioms are translated into up- and down-casting methods. This ontology API interacts with the RDF stores and the OWL/SWRL reasoner which enforces the business logic expressed in the ontology at runtime. This API provides a type-safe and familiar interface to the business logic for Java developers, keeping the reasoning hidden. The hand-written imperative code is purely technical and void of any business dimension: it is mainly a controller for REST/JSON services. The amount of hand-written code specific to these services is very small: less than 300 lines of Java code.

3.5. Integration

A semantic integration with the legacy systems has been developed thanks to an intermediation ontology which maps the 'pure' business ontology to the external data structures. A schematic of the software architecture is shown in Figure 4 below.

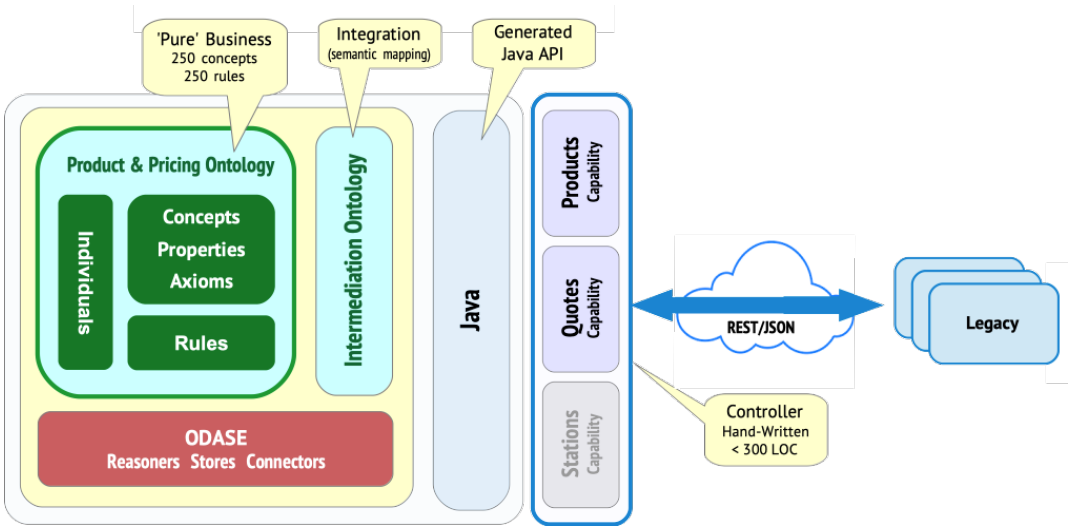
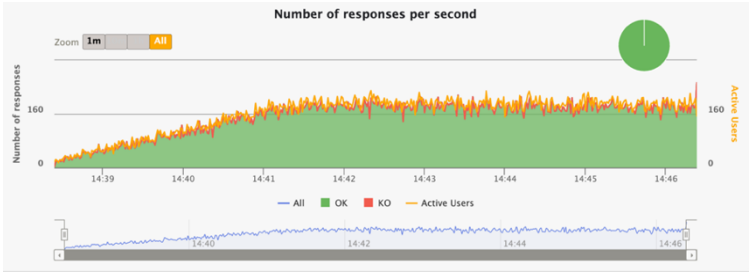


Figure 4: Software architecture

4. Performance

The application (with all the business capabilities defined ontologically), deployed in the Microsoft Azure Cloud using Kubernetes, delivers 160 rps (pricing requests per second) with 75 ms response time (95% percentile) on 4 PODs[1] with 2 cores each (see Figure 5).



[1] PODs are the smallest deployable units of computing that can be created and managed in Kubernetes. A Pod (as in a pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

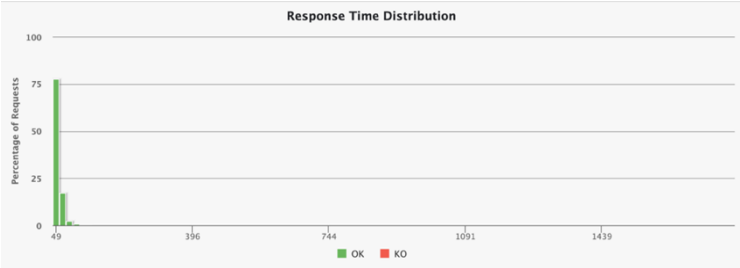


Figure 5: Throughput in pricing requests per second (number of responses per second) and response time distribution (no errors [KO] observed).

Using 8 PODs with 2 cores each delivers 350 rps with the same response time. This demonstrates a linear scale-up. This easily exceeds the original performance requirement.

These performance goals have been reached by a combination of a generic algorithm (a modified version of OLDT resolution [6]) and by what we call 'ontology engineering'.

Ontology engineering is the process of influencing the behavior of the reasoner to improve the performance. This is done by analyzing a trace of events, identifying the costliest ones, making hypothesis and adding knowledge in the form of rules. These engineering rules are technical and have no impact on the definition of the semantic of the problem. In general, this optimization process is a matter of a few days of work.

5. Effort, Timing and Correctness

This project was executed by a team of three (ontologist, software engineer and project manager), delivering a first version of correct business ontologies and rules after six months and a production-ready release in less than nine months.

The first release of the flexible season ticket system went live on March 23, 2023 and has been followed by two other major releases.

The system has had zero defects since the first release.

6. Longevity and Robustness

An ontology is independent of the technology used to implement the application it specifies (architecture, programming language, database, middleware, etc.). Being a mathematical model, it is guaranteed to outlive the implementation technology. This makes the ontology a valuable business asset in a way that traditional implementations are not: it ensures complete portability when changing hardware and software architectures. The ontology-centric approach deals with the complexity of changes with ease and robustness.

7. Discussion

The ontology-centric approach has, of course, limitations. It is not usable for hard real-time applications (due to non-determinism and garbage collection), although event-based soft real-time applications have been developed. It is also not adequate for applications requiring intensive calculus, although, when not dominant, numerical operations can be abstracted using SWRL built-ins. See example in Figure 6.

```
225 rule age-of-client
226 if ?product is a p:ProductWithClientAgeSensitivity
227     and ?person p:hasActiveSubcriptionToTravelAbo ?product
228     and ?person p:hasDateOfBirth ?birthDate
229     and ?product t:hasValidationDate ?validationDate
230     and time:ageInYears(?ageInYears, ?birthDate, ?validationDate)
231 then ?product t:clientAgeInYearsAtValidationDate ?ageInYears
232 .
```

Figure 6: Example of built-in (time:ageInYears)

8. Related Work

SBVR (Semantics of Business Vocabulary and Business Rules [7]), is an Object Management Group (OMG) specification inserted in its wider Model Driven Architecture (MDA). SBVR is a comprehensive conceptual schema for understanding what business rules are and how the meanings they express may be articulated and formalised. To our knowledge, there are no SBVR tools available to create a continuum between the specification (vocabulary and rules) and a running application, more so in real-life conditions. [8] shows how an SBVR model can be translated into an OWL+SWRL knowledge base.

Similar expressiveness extensions to SWRL, as the ones presented in 3.2, have been proposed by different researchers (see [5] for a survey). To our knowledge, they have almost never been used for practical real-life applications.

9. Future Work

We are currently developing additional tools and APIs to help subject matter experts, who are not ontologists, to modify key data by themselves, challenge the results as they like and gain a full understanding of how the ontologists understood and described the business domain and problem.

In the longer term, we wish to add fuzzy logic to the SWRL vocabulary and the reasoners in order to extend the expressivity.

10. Conclusions

The use of OWL, RDF and SWRL makes it possible to effectively separate the formal definition of the problem from the technical implementation of the solution. This is possible because SWRL rules expressing the business logic are tightly associated with the business ontology. SWRL as such is not sufficient to model real-life applications: it must be extended with aggregates, NAF and existentials. Experience shows that it is preferable to give SWRL a textual syntax that is easily understandable by the business. Finally, it is essential to use high-performance reasoners geared towards using SWRL and OWL for online business applications where data is continuously changing: classification-based reasoners should be avoided, and reasoners must exploit the parallelism offered by the multi-core CPU available.

By working in this way, a true continuum is achieved between the formal logic-based business specification and the technical construct exploiting the mainstream languages, libraries and frameworks on which all developers depend.

The *ontology-centric* approach, exemplified by this real-life use case, shows that Semantic Web Technologies bring a major contribution to software engineering. It offers a response

to Fred Brooks' observation that '*The hardest part of the software task is arriving at a complete and consistent specification, and much of the essence of building a program is in fact the debugging of the specification*" [9].

And last but not least, *"... formalizing one's knowledge in logic is often an intellectually rewarding activity, and usually reflects back on or adds insight to the problem under consideration*" [10].

Acknowledgements

We thank the business and IT teams of BRail for their contributions to the success of this project.

References

[1] https://www.belgiantrain.be/en/tickets-and-railcards/flex-abonnement

[2] OpenAPI specification https://swagger.io/specification/

[3] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al.: *Swrl: A semantic web rule language combining owl and ruleml*. W3C Member submission 21, 79 (2004); https://www.w3.org/submissions/SWRL/

[4] Somogyi, Z., Henderson, F. and Conway, T.: *The execution algorithm of Mercury, an efficient purely declarative logic programming language*. Journal of Logic Programming, 29(1–3):17–64, 1996. See also https://www.mercurylang.org/

[5] Lawan, L. and Rakib A.: *The Semantic Web Rule Language Expressiveness Extensions-A Survey*; https://arxiv.org/abs/1903.11723

[6] Tamaki, H. and T. Sato. T.: *OLD resolution with tabulation. In Proceedings of ICLP '86*, pages 84–98, July 1986.

[7] SBVR specification: https://www.omg.org/spec/SBVR/1.5/About-SBVR

[8] Ceravolo, P., Fugazza, C. and Leida, M.: *Modeling Semantics of Business Rules*. Proceedings of the 2007 Inaugural IEEE-IES Digital EcoSystems and Technologies Conference, DEST 2007. 171-176. 10.1109/DEST.2007.371965.

[9] Brooks, F.: *No Silver Bullet: Essence and Accidents of Software engineering*. In Computer, Vol. 20, No. 4 (April 1987) pp. 10-19.

[10] Sterling L. and Shapiro E.: *The Art of Prolog.: 1st edition. Advanced Programming Techniques*. MIT Press, 1986, p. xvi.