Bringing modern IDE features to Semantic Web formats with the Semantic Web Language Server

Arthur Vercruysse[1 [0000−0003−3467−9755]],
Julián Rojas Meléndez[1 [0000−0002−6645−1264]], and
Pieter Colpaert[1 [0000−0001−6917−2167]]

IDLab, Department of Electronics and Information Systems, Ghent University – imec
firstname.lastname@ugent.be

Abstract. Authoring Semantic Web documents such as ontologies or SPARQL queries is error-prone, often leading to interoperability issues, validation failures, or incorrect reasoning. To address these challenges, we present the Semantic Web Language Server (SWLS), a tool for Semantic Web practitioners that integrates IDE-like functionalities such as real-time syntax validation, intelligent autocompletion, and SHACL-based diagnostics into modern code editors. SWLS follows the Language Server Protocol, allowing seamless integration with popular code editors such as Visual Studio Code and NeoVim, while currently supporting multiple Semantic Web formats including Turtle, JSON-LD and SPARQL. This demo accompanies an accepted ESWC 2025 resource paper and showcases SWLS in an interactive web-based environment, where users can explore its features across four dedicated editor panels: (i) an RDF instance data editor, (ii) an ontology editor, (iii) a SHACL shape editor and (iv) a SPARQL query editor. The demo highlights SWLS's ability to detect syntax and semantic errors, suggest completions based on dereferenced ontologies, and assist users in writing coherent SPARQL queries. By improving the development workflow and addressing common pitfalls, SWLS aims to enhance the usability of Semantic Web technologies and facilitate broader adoption.

Keywords: Language Server, IDE, Tool, Linked Open Vocabularies, Semantic Web

URL: https://ajuvercr.github.io/semantic-web-lsp/

1 Introduction

The Semantic Web comprises a variety of syntax formats for expressing and querying data, such as Turtle, JSON-LD, and SPARQL. These formats enable representing semi-structured data in an interoperable way. However, authoring and editing documents following these formats is prone to human error, which

can lead to non-interoperability, validation failures or incorrect reasoning outcomes. For instance, mistyping the URI of a prefix can alter the entire meaning of an RDF document, rendering it non-interoperable with other datasets.

Some open services such as Linked Open Vocabularies (LOV) [3] or prefix.cc provide both human-oriented and programmatic interfaces to look up exact definitions of predicates and classes, offering a reliable reference for avoiding such errors. Tools such as YASGUI [4,6] and query.wikidata.org provide robust web-based functionalities for editing SPARQL queries. However these do not integrate into local development environments nor support multiple formats.

In this demo paper, we showcase the Semantic Web Language Server (SWLS), a language server based on the Language Server Protocol (LSP), designed to assist editors in handling Semantic Web documents. SWLS runs locally, allowing for quick development cycles and seamless integration into developer workflows, including popular code editors such as Visual Studio Code and NeoVim. Moreover, SWLS supports multiple Semantic Web formats such as Turtle, SPARQL and JSON-LD. It is also designed with extensibility in mind, making it a versatile solution. This demo paper accompanies an accepted ESWC 2025 resource paper, where more details about the SWLS architecture design can be found.

All code is licensed under the MIT License and available on GitHub[1].

2 Semantic Web Language Server

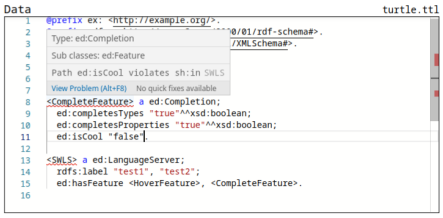
SWLS follows the Language Server Protocol (LSP) [2][2], a protocol allowing one server implementation to interact with different editors, reducing complexity from $O(M * N)$ to $O(M + N)$, with M editors and N languages.

The server communicates using JSON-RPC, enabling bidirectional messaging between the editor and server for handling features such as autocompletion, highlighting, and diagnostics. Most modern editors support LSP, including Visual Studio Code and NeoVim. The server is implemented in Rust, and allows to be compiled to WASM (portable low level byte code) that integrates into browsers.

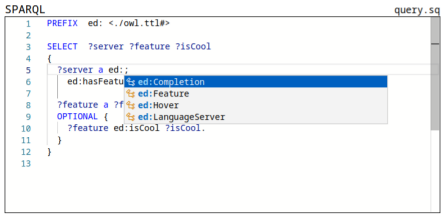
SWLS is designed for flexibility, allowing support for multiple file formats with maximal code reuse. It uses an entity component system (ECS) to build a layered architecture [1], following a traditional *Feature Execution Flow*.

1. The *Parse* step handles parsing the document (emitting potential syntax errors) and deriving defined triples and prefixes.
2. The *Traverse* step derives information from the triples, such as defined properties, classes and SHACL shapes. Starting from the defined prefixes, this step also finds the used ontologies using the public LOV api, these ontologies are added to the ECS, functioning as sources for defined properties and classes. This step also emits dignostics, notifying the editor of potential mistakes in the document, including undefined prefixes, and shape violations (using the Rudof [5] library).

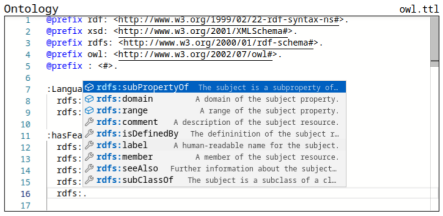
[1] `https://github.com/ajuvercr/semantic-web-lsp`
[2] `https://microsoft.github.io/language-server-protocol/`



(a) SWLS shows the user type information on hover as well as diagnostics



(b) SWLS completes a class when the user wants to write a class



(c) SWLS completes properties, first the properties with the correct domain

(d) SWLS notifies the user of undefined prefixes

Fig. 1: Demo application that shows the usage of SWLS. Features include hover information, autocompletion and diagnostics.

3. The *Process* step extracts all required information from the previous steps and translates it into a response for the current request. Currently SWLS handles autocompletion (predicates, classes and prefixes), renaming and hover.

The ECS design enables reusing the systems for the *Traverse* and *Process* steps across different languages, requiring only the *Parse* step to be implemented for each file format.

3 Demo

The demo consists of an online application at `https://ajuvercr.github.io/semantic-web-lsp/`. The app has four editor panels, each with a different purpose, and runs entirely in browser. The editors use the opensource Monaco editor[3] that communicate with the language server compiled to WASM.

1. The **data panel** represents an RDF dataset, where users can author and edit linked data. Errors detected via SHACL validation are highlighted, as shown in Figure 1a. Users can correct these errors either by adding missing triples or modifying the shape definitions in the shape panel.
2. The **ontology panel** contains a playground ontology that describes language servers. It is linked to the data panel using a prefix import, enabling autocompletion of relevant properties, as shown in Figure 1c.

[3] `https://github.com/microsoft/monaco-editor`

3. The **query panel** allows users to write SPARQL queries against the data. While SWLS does not execute queries, it provides autocompletion for classes and properties, as depicted in Figure 1b.
4. The **shape panel** includes SHACL constraints imported into the data panel via the `owl:imports` property. As shown in Figure 1d, SWLS alerts users to undefined prefixes, and typing a prefix triggers autocompletion suggestions, adding the import statement.

Users can also try the language server on local editors. For VSCode, users can install the Semantic Web LSP[4]. Setting up the server with NeoVim requires installing the language server with cargo (Rust's package manager) and configuring the language server with *lspconfig*.

4 Conclusion

The Semantic Web Language Server (SWLS) introduces IDE-like functionalities for authoring Semantic Web documents, supporting multiple formats and providing a local development workflow. By integrating features such as real-time validation, context-aware autocompletion, and SHACL-based diagnostics, SWLS lowers the chance of errors in local documents. Unlike existing tools, SWLS runs locally and extends beyond Turtle, offering a more flexible and powerful authoring experience. This work not only addresses current limitations in Semantic Web tooling but also paves the way for broader adoption of semantic technologies by reducing barriers and improving usability.

References

1. Barros, D., Peldszus, S., Assunção, W.K.G., Berger, T.: Editing support for software languages: implementation practices in language server protocols. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems. p. 232–243. MODELS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3550355.3552452, https://doi.org/10.1145/3550355.3552452
2. Bork, D., Langer, P.: Language server protocol - an introduction to the protocol, its use, and adoption for web modeling tools. Enterprise Modelling and Information Systems Architectures - International Journal of Conceptual Modeling **18**(9), 1–16 (2023). https://doi.org/10.18417/emisa.18.9
3. Dumontier, M., Vandenbussche, P.Y., Atemezing, G.A., Poveda-Villalón, M., Vatant, B.: Linked open vocabularies (lov): A gateway to reusable semantic vocabularies on the web. Semant. Web **8**(3), 437–452 (Jan 2017). https://doi.org/10.3233/SW-160213, https://doi.org/10.3233/SW-160213
4. Hyvönen, E., Rietveld, L., Hoekstra, R.: The yasgui family of sparql clients1. Semant. Web **8**(3), 373–383 (Jan 2017). https://doi.org/10.3233/SW-150197, https://doi.org/10.3233/SW-150197

[4] https://marketplace.visualstudio.com/items?itemName=ajuvercr.semantic-web-lsp

5. Labra-Gayo, J.E., Iglesias-Préstamo, A., Martín-Fernández, D., Arnaud, M.A.: rudof: A rust library for handling rdf data models and shapes (2024)
6. Rietveld, L., Hoekstra, R.: Yasgui: Not just another sparql client. In: Cimiano, P., Fernández, M., Lopez, V., Schlobach, S., Völker, J. (eds.) The Semantic Web: ESWC 2013 Satellite Events. pp. 78–86. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)