Ontology-Based Website Generation with AGAS: A DSL-Driven Approach

Diana Teixeira[1,2], José João Almeida[1,3], and Alberto Simões[1,4]

[1] Universidade do Minho, Braga, Portugal
[2] pg53766@alunos.uminho.pt
[3] jj@di.uminho.pt
[4] alberto.simoes@checkmarx.com

Abstract. AGAS is a platform that automates the generation of website prototypes from ontologies using a Domain-Specific Language (DSL). By combining semantic reasoning with customizable interface rendering, it streamlines prototyping and makes ontology-based web design more accessible.

Keywords: Automatic website generation · DSL · Ontology · Flask · SPARQL · Jinja templates · Projection editor

1 What is AGAS?

Ontologies of the non-philosophical sense[14], are known as "an explicit specification of a conceptualization."[19][12] or as tools to "capture knowledge about some domain of interest,"[15][20] and play a central role in the semantic web by enabling the formal representation of domain knowledge.

However, despite their expressive potential, ontologies are often underutilized in interface generation. Translating a structured semantic model into a user-friendly, navigable web prototype remains a challenge, especially for non-technical users.

AGAS (Application for Automatic Site Generation) is a platform designed to bridge that gap. It enables users to define how an ontology should be rendered and navigated through a custom configuration file. From that, the system generates a web interface where classes, individuals, and properties become accessible and interactive.

2 Why is it needed?

The idea for AGAS emerged from a practical need: **building website prototypes quickly based on existing domain knowledge**. In many contexts (like digital heritage, interactive storytelling, or education), data is already semantically described using TTL[8] , RDF/XML[10], and OWL[15][4], but transforming that into a web interface requires significant manual effort.

While tools such as Figma[2], Adobe XD[1], Webflow[9], Protégé/WebProtégé[25], Ontospy[7]and GraphDB[3] support design and ontology visualization, they lack mechanisms to automate the creation of interactive prototypes directly from semantic models. These platforms rarely account for ontology-encoded assets, such as media, images, or custom annotations, resulting in limited expressiveness and usability in real-world applications.

AGAS addresses these limitations through a configuration-driven, ontology-aware approach. By combining lightweight domain-specific language (DSL) files with semantic logic, the platform allows users to generate visually structured and interactive websites with minimal manual intervention. This design aims to **improve efficiency** through automation, **promote consistency and reusability** through ontology-driven structuring, and **improve usability** through adaptive knowledge-based interfaces. By reducing redundant development tasks, AGAS also supports **scalable and maintainable** prototyping workflows grounded in semantic standards.

3 What have we done so far?

AGAS is still under development, but its core architecture is already defined and implemented across three interconnected layers: configuration, processing, and interface.

At the heart of the platform lies a lightweight, declarative **Domain-Specific Language (DSL)**[23], one that, as argued by Fowler[17] and Hudak[21], serves not only as a practical tool but also as a conceptual bridge between developers and domain experts, enhancing communication [18].

This is achieved because it enables users to configure prototypes by specifying important parameters, such as:

- **Ontology Configuration**: To define the path of the ontology file, its type, and the executable location of Protégé[25], allowing seamless switching between its visual interface and a full-featured ontology editor, with live syncing or reload mechanisms planned.
- **Ontology Editing Limitations**: To specify who can edit the ontology.
- **User Experience Customization**: The DSL allows the system to tailor the website generation experience based on the user type parameter, on their level of expertise.
- **Templates Selection**: Users can specify which Jinja[5] templates should be used to render the website if they wish to use pages different than the assigned base class and individual pages. Pages with filtering, search, and navigation, implementing BREAD[13] (Browse, Read, Edit, Add, Delete) operations to promote intuitive interaction over traditional CRUD models[22] [11].
- **Visualization Preferences**: Users can define which elements should be displayed on generated pages and can even choose to use a star-based mechanism, to allow users to mark ontology **classes** and **individuals** as favorites.

On top of that, they can also choose, on one hand, which elements not to display at all and, on the other hand, which ones to prioritize in displaying.
- **RDF Visualization Preferences**: Where users can choose to see the full RDF Graph or filter its information to only display specific classes they are interested in seeing. Allowing both visualizations to appear side by side for comparative purposes.
- **Base SPARQL Queries**: In the DSL a predefined list of SPARQL[24] queries is included to extract essential ontology information. Although the system itself supports both system and user-defined queries that drive filtering, property extraction, and navigational structures.
- **Metadata Sections**: This allows the users to customize their experience, allowing them to name the ontology, to appoint information about it, to specify the disposition of lists and how they are presented (either vertically or horizontally) and even to specify which information to expand or join together in the display, as well as personal user information (emails, username, etc).

This structure allows technical and semitechnical users to control what the system does without having to manipulate the internal Python logic or HTML code. The DSL also supports default values, enabling minimal configurations that still yield functional outputs, and commenting, so users can be guided through and even document their setup themselves.

To further enhance usability, AGAS integrates a projection editor[16] powered by Monaco[6], offering a structured view of ontology content similar to Visual Studio Code, while enabling selective editing and exploration, and supports light/dark modes based on system settings or local preferences, improving accessibility across devices and environments.

4 Our Miracle Formula - From Ontology to Website

To achieve all of this and turn complex ontologies into user-friendly websites, AGAS relies on a flexible, layered system that brings together raw semantic data, a reusable component library, and a powerful configuration language.

This formula empowers users to create dynamic, personalized, and navigable ontology-based websites without ever touching low-level code. It addresses key challenges such as consistency, reusability, efficiency and scalability, all while reducing development time and improving long-term maintainability.

By combining structured knowledge, smart defaults and readable customization, AGAS transforms the way we interact with ontologies, making them not just accessible, but explorable and truly usable.

Below, we break down the three essential ingredients that make this transformation possible:

Ontology + AGAS Library + DSL.

```
Miracle Website =
```

```
Ontology (classes, taxonomy,
        properties, deduced information,
        navegation, query-language)
+
AGAS Library (default, generic and versitile
            Templates and Widgets, functions, etc)
+
DSL (
    configurate and appoint values
        Ontology Configurations
        Editing Permitions
        User Experience
        Templates Selection
        Visualization Preferences
        Graph Visualization Preferences
        SPARQL Queries
        Metadata
)
```

5 Conclusion and Future Work

AGAS demonstrates the feasibility of a low-code approach to semantic interface generation and represents a strong first step toward our goal. By combining ontologies, SPARQL, and a DSL for customization, it empowers users to prototype ontology-driven websites rapidly and meaningfully. However, like any system still in development, it has a set of known limitations and open areas for improvement.

Future development directions include:

1. **User Accounts and Access Control**;
2. **Improved Template Libraries and Examples**;
3. **Advanced Visualization Features** (more specifically timeline-based visualizations);
4. **Multi-language Support**;

References

1. Adobe xd platform - home. Adobexdplatform.com, https://adobexdplatform.com/
2. Figma: The collaborative interface design tool. Figma, https://www.figma.com/
3. Graphdb downloads and resources. Ontotext.com, https://graphdb.ontotext.com/
4. How to convert a big ontology from turtle (.ttl) format to owl/xml (.owl) format? Stack Overflow, https://stackoverflow.com/questions/76620975/how-to-convert-a-big-ontology-from-turtle-ttl-format-to-owl-xml-owl-format
5. Jinja. Pypi.org, https://pypi.org/project/Jinja2/
6. Monaco. Microsoft, https://github.com/microsoft/monaco-editor

7. Ontospy. Github.io, https://lambdamusic.github.io/Ontospy/
8. Rdf 1.1 turtle. W3C Recommendation, https://www.w3.org/TR/turtle/
9. Webflow: Create a custom website. Webflow.com, https://webflow.com/
10. Antoniou, G., Harmelen, F.V.: A Semantic Web Primer. Tlfebook (2004)
11. Şerbet Blog: Bread is the new crud. Şerbet Blog (November 12 2017), https://guvena.wordpress.com/2017/11/12/bread-is-the-new-crud/
12. Calero, C., Ruiz, F., Piattini, M.: Ontologies for Software Engineering and Software Technology. Springer (7 2006)
13. Chung, T.: Bread vs crud. GitHub Pages
14. Corcho, O., Fernández-López, M., Gomez-Perez, A.: Ontological engineering: Principles, methods, tools and languages. Ontologies for Software Engineering and Software Technology (01 2006). https://doi.org/10.1007/3-540-34518-3_1
15. Debellis, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C., Jupp, S., Moulton, G., Drummond, N., Brandt, S.: A practical guide to building owl ontologies using protégé 5.5 and plugins (2021), https://www.researchgate.net/publication/351037551
16. Fowler, M.: Projectional editing. Martinfowler.com (2008), https://martinfowler.com/bliki/ProjectionalEditing.html
17. Fowler, M.: Domain-Specific Languages. Addison-Wesley (2010)
18. Fowler, M., Spinellis, D.: A pedagogical framework for domain-specific languages. IEEE Software (2009)
19. Gruber, T.R.: A translation approach to portable ontology specifications (1993)
20. Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C., Jupp, S., Moulton, G., Drummond, N., Brandt, S.: A practical guide to building owl ontologies using protégé 4 and co-ode tools edition 1.3 (2011)
21. Hudak, P.: Domain specific languages * (1997)
22. Jones, P.M.: Bread, not crud (2008), https://paul-m-jones.com/post/2008/08/20/bread-not-crud/
23. Lethrech, M., Kenzi, A., Elmagrouni, I., Nassar, M., Kriouile, A.: A process definition for domain specific software development. IEEE (2015)
24. Prud'hommeaux, E., Seaborne, A.: Sparql query language for rdf. W3C Recommendation (2013), https://www.w3.org/TR/rdf-sparql-query/
25. Stanford: Protégé. Stanford.edu, https://protege.stanford.edu/