OWLschemafy – A Java-based, Text-Driven Ontology Generator for Streamlined OWL

Edit Hlaszny, PhD
Dr Hlaszny Bioystems Engineering
Mail: edit@edithlaszny.eu
http://www.edithlaszny.eu/

Abstract

OWLschemafy presents a streamlined ontology development approach through a user-friendly text-based interface. This application extracts its operational directives from the user-generated text files, fostering the creation of OWL/XML encoded ontologies. Furthermore, OWLschemafy integrates with the Protégé Ontology Editor, permitting users to capitalize on the strengths of both applications.

Keywords

Ontology; OWL generator; front end for Protégé; Java-based application.

1. Introduction

1.1 Ontologies: The Bedrock Of Knowledge Representation

In the ever-expanding digital landscape, the ability to effectively organize and represent knowledge is paramount [5][7]. Ontologies play a critical role in this endeavor, acting as formal specifications of a domain's concepts, properties, and relationships. The importance of ontologies lies in their ability to:

Standardize Knowledge Representation:

Ontologies provide a common vocabulary and structure for describing entities within a domain [9]. This enhance clear communication and reduces ambiguity when dealing with complex knowledge.

Enable Knowledge Sharing and Reuse:

By establishing a shared understanding, ontologies facilitate the seamless exchange of knowledge between different systems and applications. Existing ontologies can be reused and extended, promoting efficiency and avoiding the duplication of effort.

Support Reasoning and Inference:

Ontologies encode logical relationships between concepts, enabling machines to perform automated reasoning and draw inferences from the encoded knowledge. This capability underpins various applications, such as information retrieval, decision support systems, and knowledge base completion [9].

1.2 Challenges In Traditional Ontology Development

Despite their undeniable benefits, traditional ontology development methods often face several challenges:

Knowledge Acquisition Bottleneck:

The initial process of identifying, defining, and formalizing domain knowledge can be time-consuming and resource-intensive. This "knowledge acquisition bottleneck" presents a significant hurdle in ontology development.

Limited User-Friendliness:

Traditional methods often rely on specialized tools and expertise, making them less accessible to users without a background in ontology engineering [2].

Scalability Issues:

As ontologies grow in size and complexity, managing and maintaining them using traditional methods can become increasingly difficult.

These challenges highlight the need for innovative approaches that streamline the ontology development process, increase userfriendliness, and ensure scalability for larger knowledge structures.

2. OWLschemafy Is User-Centric

2.1 Embracing Text-Driven Control For Streamlined Owl Development

Traditional ontology development methods often involve intricate tools and knowledge of ontology languages like OWL (Web Ontology Language). This chapter introduces OWLschemafy, a text-driven ontology generator that empowers users to create OWL ontologies through a user-friendly and intuitive interface text files.

2.2 The Power Of Text-Based Control

OWLschemafy leverages the flexibility and accessibility of text files to provide users with granular control over the ontology generation process. These text files act as the building blocks for constructing OWL ontologies, specifying various elements through a clear and concise syntax. The key advantages of using text files for ontology control include

Enhanced Readability and Editability:

Text files are readily human-readable and editable using any basic text editor. This user-friendliness lowers the barrier to entry for ontology development, making it accessible to a broader range of users without requiring specialized knowledge of OWL syntax.

Flexibility and Customization:

Text files offer a high degree of flexibility in specifying the ontology structure and content. Users can tailor the control files to their specific needs, defining elements like classes, properties, and relationships in a way that aligns with their understanding of the domain.

Version Control and Collaboration:

Text files are inherently version-controllable, enabling users to track changes and revert to previous versions if necessary. This feature facilitates collaboration, allowing multiple users to work on the same ontology by editing and sharing the control files.

2.3 Demystifying The Control File Structure

OWLschemafy employs a well-defined structure within the text files to capture the various components of an OWL ontology [7]. Key elements specified in these files include

OWL Class Structure:

The hierarchical structure of classes within the ontology is defined using indentation. Each level of indentation represents a subclass relationship, adhering to the tab-based indentation convention compatible with Protégé, a widely used ontology editor.

Class Disjointness Definitions:

These files allow users to specify classes that are mutually exclusive, ensuring that no individual can belong to both classes simultaneously.

Class-Individual Definitions:

Individuals, representing specific instances within a class, are defined using text files. These definitions can include data

property values and object property relationships, linking individuals to other entities within the ontology.

Entity Annotations:

OWLschemafy empowers users to attach annotations to various entities (classes, individuals, properties) within the ontology. These annotations enrich the ontology with additional information, such as comments, definitions, or references to external resources.

Triplet Generation:

Arguably the most crucial aspect of the control files lies in their ability to define triplets. Triplets, the fundamental building blocks of OWL ontologies, represent relationships between entities. Text files specify these triplets, dictating how classes, properties, and individuals are interconnected.

By leveraging this intuitive text-based approach, OWLschemafy empowers users to construct rich and expressive OWL ontologies without grappling directly with the complexities of OWL syntax. The following sections will delve deeper into the core functionalities of OWLschemafy, outlining how it streamlines the ontology development process for users of all backgrounds.

3. OWLschemafy is Text-Driven

3.1 Enhancing User-Centricity With Text-File Control

This chapter seeks into the core functionalities of OWLschemafy, highlighting its innovative approach to ontology development. OWLschemafy departs from the traditional paradigm of directly writing OWL code, instead leveraging the user-friendliness and accessibility of text files as the primary control mechanism. This user-centric design philosophy empowers researchers and developers to construct OWL ontologies through an intuitive and streamlined process.

The Figure 1. below visually portrays the interplay between OWLschemafy and Protégé.

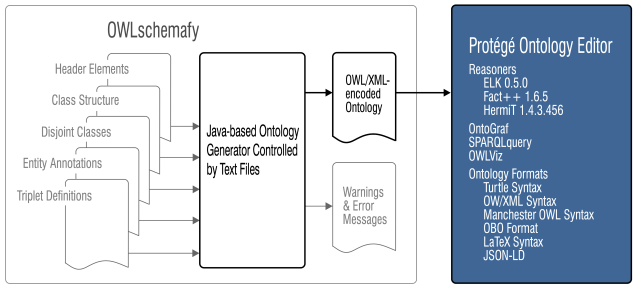


Figure 1: OWLschemafy as front end to Protégé.

OWLschemafy acts as a front end, utilizing user-provided text files to generate OWL/XML encoded ontologies: These generated ontologies can then be seamlessly imported and further refined within the familiar environment of Protégé, a widely adopted ontology editor [6][10].

3.2 Key Functionalities For Streamlined Development

OWLschemafy offers a comprehensive suite of functionalities geared towards streamlining the ontology development lifecycle. Here's a glimpse into some of its key features:

4. Key Functionalities Of OWLschemafy

4.1 Text-File Driven Control – A User-Friendly Interface Vs. Its Difficulties

OWLschemafy's cornerstone lies in its text-file driven control mechanism. Users create and edit text files with a well-defined structure, specifying the various components of the ontology. However, the difficulties arising from the use of text files cannot be ignored, such as

Scalability:

As ontologies grow larger and more complex, text-based control files can become cumbersome to manage. Maintaining consistency and tracking changes across numerous files can be challenging.

Error-prone:

Manually editing text files increases the risk of introducing errors during definition or modification. Version control becomes crucial for managing changes and reverting to previous states.

Limited Functionality:

Text files offer limited functionalities for complex control structures or advanced operations. Complex logic or dependencies within the control flow might be difficult to represent effectively.

Querying and Analysis:

Analyzing and querying information within the control files can be challenging. Extracting specific details or performing comparisons across different files requires manual effort or custom scripting.

4.2 Controlling Ontology Generation With Text Files In OWLschemafy

OWLschemafy uses five distinct text files to exert granular control over the ontology generation process. These files address specific aspects of the ontology structure and content, adhering to a well-defined sequence that mirrors the ontology's construction.

Ontology Header File:

The initial text file defines the header information for the generated ontology, including essential metadata elements.

OWL Class Structure File:

This file specifies the hierarchical organization of classes within the ontology. A well-defined indentation scheme, often adhering to tab-based conventions for compatibility with Protégé, represents subclass relationships.

Class Disjointness File:

OWLschemafy enables the specification of disjoint classes through a dedicated text file. Disjoint classes are mutually exclusive categories, ensuring that no individual can belong to both classes simultaneously.

Class Annotation File:

Annotations enrich the ontology by providing additional information about classes, such as definitions, comments, or references to external resources. A dedicated text file facilitates the incorporation of such annotations.

Triplet Definition File:

Arguably the most intricate aspect of ontology development lies in defining triplets, the fundamental building blocks that express relationships between entities. The final text file within OWLschemafy's control mechanism caters to this critical task. It encompasses not only the specification of triplets but also the creation of the necessary named individuals, the object-, and data-properties that participate in these relationships.

This sequential order of text file processing reflects the step-by-step construction of the ontology itself. By meticulously crafting these control files, users empower OWLschemafy to generate ontologies tailored to their specific needs [1].

4.3 Streamlined OWL/XML Generation – Automation For Efficiency

Once users have meticulously crafted their control files, the OWLschemafy automates the generation of OWL/XML encoded ontologies. This automation significantly reduces the time and effort required for ontology development compared

to manual OWL code writing. The core functionalities involved in the generation process include:

Parsing Control Files:

OWLschemafy parses the user-provided text files, extracting the specified ontology elements and their relationships.

Generating OWL/XML Syntax:

Based on the parsed information, OWLschemafy translates the extracted elements and relationships into the corresponding OWL/XML syntax, adhering to the OWL standard.

Validation and Error Checking:

OWLschemafy incorporates functionalities to validate the user-provided information within the control files. This ensures that the generated OWL/XML output adheres to the OWL syntax and avoids potential errors.

4.4 Protégé Integration – A Synergistic Workflow

OWLschemafy empowers a synergistic workflow by seamlessly integrating with Protégé, a widely adopted ontology editor. This integration offers several benefits:

Leveraging Protégé's Visual Editing Capabilities:

The generated OWL/XML ontologies can be effortlessly imported into Protégé. Once imported, users can inflict Protégé's visual editing interface to further refine and edit the ontology structure. This visual representation can be particularly helpful for complex ontologies.

Comprehensive Ontology Management:

Protégé offers a suite of functionalities for managing ontologies, including support for reasoning, version control, and collaboration. This integration allows users to seamlessly switch between text-based control in OWLschemafy and Protégé's comprehensive ontology management environment.

4.5 Protégé's Functionalities Complementing OWLschemafy's Workflow

While OWLschemafy acts as a user-centric front end for ontology development, its efficacy is significantly amplified when coupled with the comprehensive functionalities offered by Protégé, a widely adopted ontology editor. Protégé provides a rich set of features that seamlessly integrate with OWLschemafy's output, empowering users to further refine, analyze, and manage their ontologies.

Reasoning Capabilities:

Protégé integrates several reasoners out-of-the-box, including ELK 0.5.0, Fact++ 1.6.5, and HermiT 1.4.3.456. These reasoners enable users to assess the logical consistency and infer implicit knowledge within their ontologies, facilitating the creation of robust and well-founded knowledge structures.

Visualization and Navigation:

Protégé's OntoGraf plugin facilitates the visualization of specific ontology subgraphs, allowing users to focus on intricate portions of the knowledge structure. Additionally, the OWLViz plugin empowers users to navigate and explore class hierarchies within the ontology.

Querying and Export:

Protégé offers a SPARQL query engine, enabling users to formulate queries and retrieve specific information from the ontology. This capability fosters in-depth exploration and analysis of the encoded knowledge.

Leveraging these complementary functionalities within Protégé, users can effectively extend the capabilities of OWLschemafy, enriching the ontology development process and empowering the creation of comprehensive and well-reasoned knowledge structures.

Intuitive Text-File Based Control:

As mentioned earlier, OWLschemafy leverages text files for user control. These files house detailed specifications for the ontology structure, including classes, properties, relationships, and annotations. The straightforward syntax employed within the text files makes them readily understandable and editable using standard text editors.

Streamlined OWL/XML Generation:

Once users have meticulously crafted their control files, OWLschemafy automates the generation of OWL/XML encoded ontologies. This automation significantly reduces the time and effort required for ontology development compared to manual OWL code writing.

Protege Integration:

OWLschemafy supports a synergistic workflow by seamlessly integrating with Protégé. The generated OWL/ XML ontologies can be effortlessly imported into Protégé, allowing users to take advantage its visual editing capabilities and comprehensive ontology management functionalities.

By harnessing these core functionalities, OWLschemafy empowers users to construct OWL ontologies with greater efficiency, flexibility, and user-friendliness. The subsequent sections will delve deeper into specific functionalities, illustrating how OWLschemafy simplifies various aspects of the ontology development process.

5. Technical Details And Implementation

5.1 Beyond Functionality: A Deep Dive

This paper investigates beyond introducing a novel front end for Protégé. It offers a comprehensive technical description, meticulously detailing the control files (user interaction building blocks) and the core Java source code. Detailed JavaDoc documentation further enhances understanding and future development.

This transparency empowers researchers and developers to replicate the software and contribute to its growth. The readily available JavaDoc streamlines future modifications and reduces the learning curve.

By extending its scope, this paper establishes a strong foundation for further exploration and development of this technical framework.

To showcase the capabilities of OWLschemafy, this chapter looks into modeling a relatively simple and well-defined, compact (that is bounded and closed :) domain: the PRINCE2 Project Management System. Effectively modeling a technical subject area necessitates three key elements:

- *In-depth Domain Knowledge:* A comprehensive understanding of the chosen domain is essential.
- *Modeling Expertise:* A solid grasp of modeling principles and methodologies is crucial.
- *Suitable Modeling Tool:* The selection and proficient use of an appropriate modeling tool streamlines the process.

Here, we focus on the third element, specifically the Protégé ontology editor enhanced by OWLschemafy, the front-end tool introduced in this paper.

5.2 Ontology Design Considerations

Developing ontologies necessitates careful consideration of several key principles:

Top-Down Approach:

Ontologies adhere to a top-down design philosophy, commencing with a conceptualization of the domain and progressing towards the electronic management and storage of information.

Reusability and Interoperability:

A well-designed ontology should promote reusability across diverse domains. For instance, the generated OWL file can be

seamlessly integrated into a SPARQL-enabled graph database (e.g. Nebugraph, Neo4j, Ontotext GraphDB, or Oracle RDF Graph; part of Oracle Database), fostering interoperability and knowledge exchange [3][4]. Ontotext GraphDB can directly load the generated OWL/XML encoded ontology.

Accommodating Domain Growth:

The ontology should be adaptable to accommodate the influx of new information continuously generated within the domain it represents. This ensures the ontology's continued relevance and utility over time. A flexible ontology structure, accompanied by clear extension mechanisms, is essential for managing domain growth without compromising the ontology's integrity.

Preserving Existing Synergies:

Maintaining and potentially expanding established synergies and interfaces with other tools is crucial. This facilitates a more integrated knowledge management ecosystem.

5.3 Ontology Header Specification

An ontology typically commences with a standardized header. This header incorporates essential information such as the XML version, xmlns definitions, prefix names, and annotations. Common annotations include dc:title, dc:author, dc:date, and other DCMI metadata terms, such as language, contributor, version, phone number.

OWLschemafy uses a dedicated control file, `P2_HeaderDef.txt` the content of this control file, which consists of `IRI` and `LITERAL` pairs. Empty lines and lines starting with "!" are ignored by OWLschemafy. "!" allows for comments within the files.

The ontology header specificationsis depicted on the Figure 2.

_HeaderDef.txt

```
!+
!   MODIFICATION HISTORY:
!     date        modified by
!     25-JUN-2024 Hlaszny  1-001 first draw
!     03-JUL-2024 Hlaszny  1-002 dc.date actualized
!-

IRI=dc:title
LITERAL=PRINCE2 - Ontology of PRINCE2 Management
System

IRI=dc:description
LITERAL=The subject of the ontology is the PRINCE2

IRI=dc:rights
LITERAL=This ontology is distributed under an
Attribution 4.0 International (CC BY 4.0) License -
https://creativecommons.org/licenses/by/4.0/.
Copyright: Dr. Hlaszny Biosystems Engineering (
2024-)

IRI=dc:publisher
LITERAL=Dr. Hlaszny Biosystems Engineering: In Vivo
Integration of Neural Parenchyma

IRI=dc:creator
LITERAL=Edit Hlaszny
>
IRI=dc:contributor
LITERAL=I owe special thanks to Dr Henriette Harmse
(European Bioinformatics Institute) and to Dr Marie
Keet (University of Cape Town) for their versatile
advice and encouragement.
```

Figure 2: Ontology header definition.

OWLschemafy translates the content of the aforementioned control file into the following OWL code snippet on Figure 3.

prince2_ontology.owl

```
<Annotation>
    <AnnotationProperty abbreviatedIRI="dc:title"/>
    <Literal>PRINCE2 - Ontology of PRINCE2 Management System
    </Literal>
</Annotation>
<Annotation>
    <AnnotationProperty abbreviatedIRI="dc:description"/>
    <Literal>The subject of the ontology is the PRINCE2</Literal>
</Annotation>
<Annotation>
    <AnnotationProperty abbreviatedIRI="dc:rights"/>
    <Literal>This ontology is distributed under a Attribution
    4.0 International (CC BY 4.0) License -
    https://creativecommons.org/licenses/by/4.0/. Copyright: Dr.
    Hlaszny Biosystems Engineering (2024-)</Literal>
</Annotation>
<Annotation>
    <AnnotationProperty abbreviatedIRI="dc:publisher"/>
    <Literal>Dr. Hlaszny Biosystems Engineering: In Vivo
    Integration of Neural Parenchyma</Literal>
</Annotation>
<Annotation>
    <AnnotationProperty abbreviatedIRI="dc:creator"/>
    <Literal>Edit Hlaszny</Literal>
</Annotation>
<Annotation>
    <AnnotationProperty abbreviatedIRI="dc:contributor"/>
    <Literal>I owe special thanks to Dr Henriette Harmse (
    European Bioinformatics Institute) and to Dr Marie Keet (
    University of Cape Town) for their versatile advice and
    encouragement.
    </Literal>
</Annotation>
```

Figure 3: Genereated OWL/XML code as ontology header by OWLschemafy.

Figure 4. showcases the OWL/XML encoded code displayed within the Protégé environment. While the default order of generated OWL ontology items in Protégé does not match the original ontology header, this behavior can be modified to reflect the sequence of scanned header items.

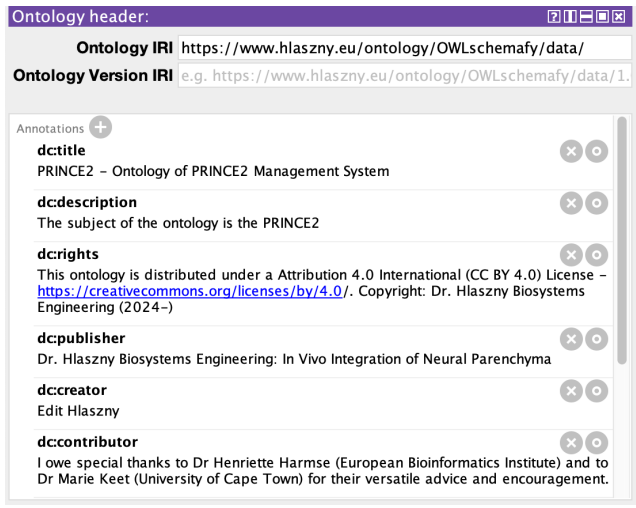


Figure 4: Ontology header displayed in Protégé.

5.4 Controlled Vocabulary And Class Definition:

The practical process of ontology construction often begins with the creation of a simplified vocabulary. The simplified vocabulary (Figure 5) serves as the definitive foundation for an ontology because it establishes the core building blocks for representing knowledge within a specific domain.

It defines the essential concepts, their relationships, and the attributes that describe them. Just as a building relies on a strong foundation, a robust and well-defined simplified vocabulary ensures the clarity, interoperability, and reasoning capabilities of an ontology.

Simplified PRINCE2 Vocabulary

acceptance

- **Basically:** The official thumbs up that the project delivered what everyone wanted.

acceptance criteria

- **Think of it as a checklist:** The project isn't finished until everything on this list is checked off. This ensures it meets everyone's needs.

activity

- **Just a fancy word for a task:** Something that takes time, has a clear outcome, and needs to be managed. Think of it as a step in the project plan.

agile methods

- **Imagine building software in small pieces:** Agile methods do that, delivering working features bit by bit. PRINCE2 can work alongside these methods.

approval

- **Green light!** Someone (usually a designated group) officially says the project output is finished and meets the requirements, with maybe a few minor tweaks.

approver

- **The person (or group) in charge:** They have the final say on whether a project piece is complete and ready to move forward. Think of them as the final checkpoint.

assumption

- **A best guess:** We're making a plan based on something we believe to be true, but it might change later. If that happens, we might need to adjust the plan.

assurance

- **Making sure everything's on track:** It's about having confidence that the project is heading in the right direction and will deliver what it's supposed to. Think of it as a double-check.

Figure 5: Simplified PRINCE2 Vocabulary.

Protégé uses a well-defined tab-based indentation scheme to represent class hierarchies, a convention that OWLschemafy faithfully preserves. From the tab-delimited control file (`P2_classStructure.txt`) illustrates a part the Figure 6. This preservation facilitates direct comparison within Protégé, particularly for intricate or ambiguous class relationships.

P2_classStructure.txt

```
PRINCE2
    Actor
        Board
            ProjectExecutive
```

Figure 6: Tab-indented configuration file defines the class-structure of the ontology.

prince2_ontology.owl

```
<Declaration>
    <Class IRI="http://.../OWLschemafy/data/#PRINCE2"/>
</Declaration>
<Declaration>
    <Class IRI="http://.../OWLschemafy/data/#Actor"/>
</Declaration>
<Declaration>
    <Class IRI="http://.../OWLschemafy/data/#Board"/>
</Declaration>
<Declaration>
    <Class IRI="http://.../OWLschemafy/data/#ProjectExecutive"/>
</Declaration>
```

Figure 7: Class-structure in the ontology generated by OWLschemafy.

prince2_ontology.owl

```
<SubClassOf>
    <Class IRI="http://.../OWLschemafy/data/#Actor"/>
    <Class IRI="http://.../OWLschemafy/data/#PRINCE2"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="http://.../OWLschemafy/data/#Board"/>
    <Class IRI="http://.../OWLschemafy/data/#Actor"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="http://.../OWLschemafy/data/#ProjectExecutive"/>
    <Class IRI="http://.../OWLschemafy/data/#Board"/>
</SubClassOf>
```

Figure 8: Subclass definitions in the ontology generated by OWLschemafy.

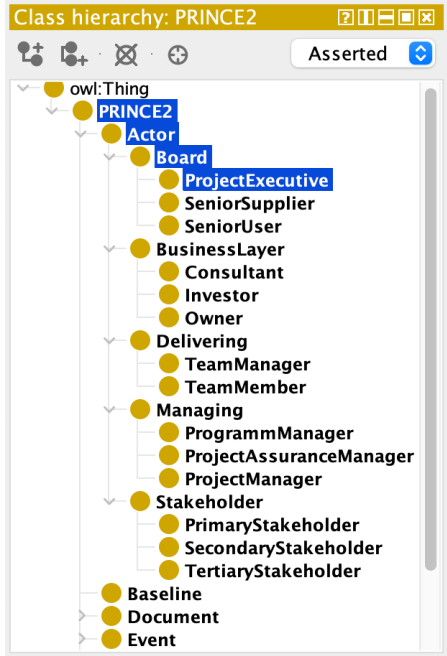


Figure 9: Class-structure in Protégé.

Figures 10, 11, and 12 illustrate the relationship between the class annotations contained control file `P2_classAnnotation.txt`, the corresponding OWL/XML code snippet generated by OWLschemafy, and the resulting visualized within Protégé:

P2_classAnnotation.txt

```
CLASS_NAME       = PRINCE2
CLASS_ANNOTATION = "PRINCE2 (Projects IN
Controlled Environments) is a project management
methodology designed to ensure effective project
delivery within a structured framework.  It
emphasizes a stage-gate process, dividing the
project lifecycle into well-defined phases with
clear deliverables and decision points.  PRINCE2
promotes a balance between project justification,
business alignment, risk management, and
continual justification throughout the project
lifecycle.  It provides a comprehensive framework
for project planning, execution, and control,
making it a widely adopted methodology across
various industries."
```

Figure 10: Class annotation definition in the control file.

prince2_ontology.owl

```
Scroll Tabs
1540    <AnnotationAssertion>
1541        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
1542        <IRI>http://.../OWLschemafy/data/#PRINCE2</IRI>
1543        <Literal>"PRINCE2 (Projects IN Controlled Environments)
1544                  is a project management methodology designed
1545                  to ensure effective project delivery within
1546                  a structured framework.  It emphasizes a
1547                  stage-gate process, dividing the project
1548                  lifecycle into well-defined phases with clear
1549                  deliverables and decision points. PRINCE2
1550                  promotes a balance between project
1551                  justification, business alignment, risk
1552                  management, and continual justification
1553                  throughout the project lifecycle.  It
1554                  provides a comprehensive framework for
1555                  project planning, execution, and control,
1556                  making it a widely adopted methodology
1557                  across various industries."
1558        </Literal>
    </AnnotationAssertion>
```

Figure 11: Generated class annotation in the generated `.owl` file.

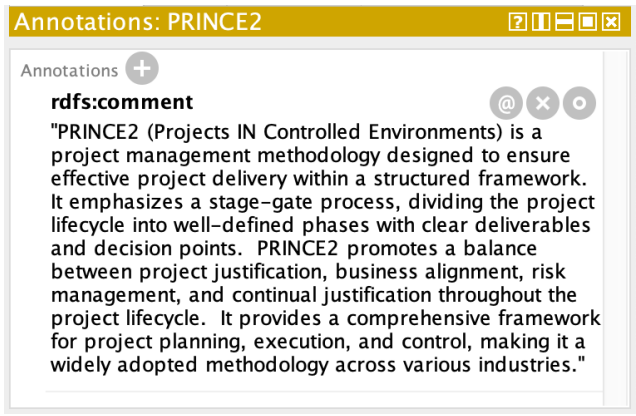


Figure 12: Class annotation displayed in Protégé.

5.5 Disjoint Classes In Ontologies

There are several key considerations when determining whether to define classes as disjoint.

Mutual Exclusivity and Domain Specificity:

The primary consideration is whether membership in one class inherently eliminates the possibility of belonging to the other. The decision to make classes disjoint can be domain-specific.

Reasoning and Inference:

Declaring classes as disjoint can enhance the reasoning capabilities of an ontology. By explicitly stating that certain classes are mutually exclusive, the ontology reasoner can identify inconsistencies and draw more accurate inferences based on the encoded knowledge.

OWLschemafy facilitates the definition of disjoint classes through a dedicated control file. This file explicitly specifies which classes within the ontology should be mutually exclusive. Figures 13, 14, and 15 illustrate the interrelated aspects of this process: the control file (`P2_DisjointClassDefs.txt`) containing the disjoint class definitions, the corresponding OWL/XML code snippet generated by OWLschemafy based on these definitions, and the resulting visualization within Protégé's DL Class Axiom Renderer.

Ontology Structure and Maintainability:

This is a crucial factor when deciding on disjoint classes. The ontology designer should be carefully considered how disjoint classes impact the overall ontology structure, how can enhance maintainability and prevent inconsistencies.

P2_DisjointClassDefs.txt

```
762  START_OF_DISJOINT_CLASS_SET
763      Project
764          Benefit
765          Outcome
766          Output
767          Product
768          Risk
769          Stage
770              PreprojectStage
771              InitiationStage
772              SubsequentStage
773              FinalStage
774              PostprojectStage
775  END_OF_DISJOINT_CLASS_SET
```

Figure 13: Disjoint class definitions in the control file.

prince2_ontology.owl

```
3661  <DisjointClasses>
3662      <Class IRI="http://.../OWLschemafy/data/#PreprojectStage"/>
3663      <Class IRI="http://.../OWLschemafy/data/#InitiationStage"/>
3664      <Class IRI="http://.../OWLschemafy/data/#SubsequentStage"/>
3665      <Class IRI="http://.../OWLschemafy/data/#FinalStage"/>
3666      <Class IRI="http://.../OWLschemafy/data/#PostprojectStage"/>
3667  </DisjointClasses>
3668  <DisjointClasses>
3669      <Class IRI="http://.../OWLschemafy/data/#Benefit"/>
3670      <Class IRI="http://.../OWLschemafy/data/#Outcome"/>
3671      <Class IRI="http://.../OWLschemafy/data/#Output"/>
3672      <Class IRI="http://.../OWLschemafy/data/#Product"/>
3673      <Class IRI="http://.../OWLschemafy/data/#Risk"/>
3674      <Class IRI="http://.../OWLschemafy/data/#Stage"/>
3675  </DisjointClasses>
```

Figure 14: Generated disjoint class definitions in the result `.owl` file.

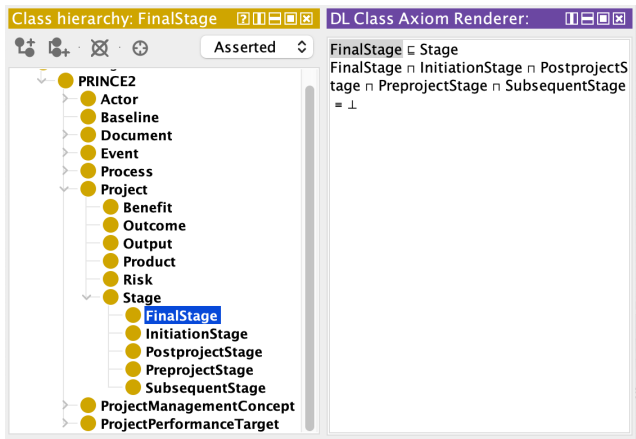


Figure 15: Disjoint classes' visualization within Protégé's DL Class Axiom Renderer.

6. Triplet Definitions In OWLschemafy

The Protégé front end OWLschemafy employs a specific format (`P2_tripletDefs.txt`) for defining triplets (subject, predicate, and object), which represent relationships between entities within the ontology in which the following elements are involved:

6.1 Comments and Separators:

Lines in each control files starting with "!+" and "!-" are comments (or even block comments, which can be extracted as a whole) used to enhance readability and separate sections

within the definition. Empty lines will also be negligated.

6.2 Triplet Set:

The definition starts with `START_OF_TRIPLE_SET` and ends with `END_OF_TRIPLE_SET`, demarcating the entire set of triplets being defined.

6.3 Header Section:

`START_OF_HEADER` and `END_OF_HEADER`: these markers enclose the header information for the triplet set. This header defines various properties associated with the relationships specified in the triplets.

`PREDICATE_NAME` and `INV_PREDICATE_NAME`: These define the object property names used to represent the relationship in both directions. For example, `documentCreatedBy` and `createsDocument` represent the relationship between a Document and an Actor (who created it).

`SUPER_OBJECT_PROPERTY`: This specifies the superclass of the object property in the OWL hierarchy (e.g., owl:topObjectProperty).

6.4 Triplet Definitions:

This section defines the actual relationships between entities. Figure 16. shows a triplet set. Each line after the triplet set header (from the line 925) represents a single triplet.

P2_tripletDefs.txt

```
!+
!   Relationship:  Document <creating/using> Actor
!-
START_OF_TRIPLET_SET

START_OF_HEADER

PREDICATE_NAME                  = documentCreatedBy
INV_PREDICATE_NAME              = createsDocument
SUPER_OBJECT_PROPERTY           = owl:topObjectProperty
OBJECT_PROPERTY_ANNOTATION      = "The object property 'documentCreatedBy'
                                  encodes which Document is created by whom."
INV_OBJECT_PROPERTY_ANNOTATION  = "The object property 'createsDocument'
                                  encodes which Actor creates which Document."

SUBJECT_DATA_PROPERTY_NAME      = documentAsActorsContribution
SUBJECT_DATA_PROPERTY_TYPE      = String
SUBJECT_DATA_PROPERTY_ANNOTATION = "The data property
                                  'documentAsActorsContribution' identifies
                                  the portion created by an actor."

OBJECT_DATA_PROPERTY_NAME       = actorsEffortAtDocumentCreation
OBJECT_DATA_PROPERTY_TYPE       = decimal
OBJECT_DATA_PROPERTY_ANNOTATION  = "The data property
                                  'actorsEffortAtDocumentCreation' describes
                                  the contribution of an actor."

SUPER_DATA_PROPERTY             = owl:topDataProperty

END_OF_HEADER

!+
!   subject    documentCreatedBy   object            subject         object
!                                                   data propery    data propery
!-
ProjectInitiationDoc            ProjectExecutive    MEDIUM          0.35
ProjectInitiationDoc            SeniorUser          LOW             0.15
ProjectInitiationDoc            Owner               MEDIUM          0.2
ProjectInitiationDoc            ProjectManager      HIGH            0.3
ProjectBrief                    ProjectExecutive    HIGH            0.5
ProjectBrief                    SeniorUser          LOW             0.1
ProjectBrief                    Owner               MEDIUM          0.4
!
END_OF_TRIPLET_SET
```

Figure 16: Triplet Set Definitions.

- *Subject and Object:* These represent the OWL class-instances (individuals) of the classes participating in the relationship.
- *Data Property Values:* Following the subject and object, there can be additional values associated with the data properties defined in the header.
- *Class individuals:* OWLschemafy can automatically generate class individuals (e.g., with sequential numbering) when triplets are defined. This eliminates the need to predefine every single instance within the ontology.
- *Data properties:* are also generated based on the information provided in the triplet set header. This streamlines the definition process.

Overall, triplet definitions in OWLschemafy offer a concise and efficient way to specify relationships and their associated data properties within an ontology. The header section provides context and meaning for the relationships, while the individual triplets establish the specific instances and their corresponding values according the OWL language definitions [7].

6.5 Steps Containing Triplet Set Definitions:

The following ten steps are needed to define triplets

1. *Object Property Declaration:* OWLschemafy declares two object properties based on the information in the header. The first property represents the primary direction of the relationship, and the second property represents its inverse.
2. *Inverse Object Property Relationship:* OWLschemafy establishes that the two declared object properties are inverses of each other. This ensures that the relationship can be navigated in both directions within the ontology.
3. *Object Property Annotations:* OWLschemafy incorporates human-readable annotations for both object properties. These annotations provide context and clarify the meaning of the relationships being defined.
4. *Super Object Property:* OWLschemafy assigns a superclass (e.g., `owl:topObjectProperty`) to the declared object properties within the OWL hierarchy. This helps categorize and organize the properties within the ontology.
5. *Subject and Object Class Individuals:* OWLschemafy can automatically generate instances (individuals) of the classes participating in the relationship. These instances represent specific entities within the domain.
6. *Data Property Declaration:* OWLschemafy declares data properties associated with the subject and object of the relationship. These properties provide additional details or attributes about the entities involved.
7. *Data Property Annotations:* Similar to object properties, annotations are added to data properties to explain their purpose and meaning.
8. *Super Data Property:* A superclass (e.g. `owl:topDataProperty`) is assigned to the data properties within the OWL hierarchy for organization.
9. *Connecting Data Properties and Individuals:* OWLschemafy establishes connections between the data properties and the class individuals participating in the relationship. This allows associating specific values with the entities.
10. *Triplet Definition:* Finally, OWLschemafy defines the actual triplets based on the object properties and the class individuals. Each triplet represents a specific relationship instance between two entities. Due to space constraints, presenting each step with its corresponding code snippet is beyond the scope of this paper. However, these details are available within the `prince2_ontology.owl` file, which represents the PRINCE2 ontology. Therefore, this paper will focus on the final results visualized within the Protégé environment.

Figure 16 illustrates a triplet set. The 925th line specifies relationship between classes. Based on this information in the control file, OWLschemafy generates the corresponding OWL/XML encoded ontology. This ontology can then be visualized within the Protégé environment, as shown in the Figure 17.

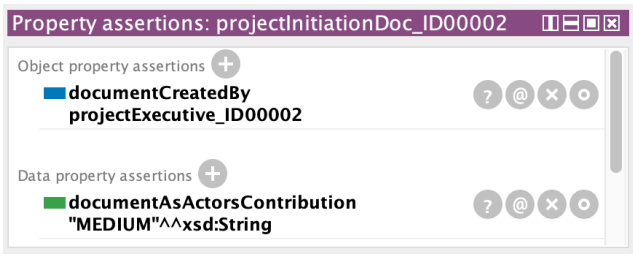


Figure 17: Triplet Definitions visualised in Protégé.

The Figure 17 shows the `documentCreatedBy` object property within the Protégé environment. It represents the relationship between the class individuals `ProjectInitiationDoc` and `ProjectExecutive` as follows:

- *Subject and Object:*
  These represent the instances (individuals) of the classes participating in the relationship.
- *Object Property:*
  The label `documentCreatedBy` is displayed, indicating the name of the object property.
- *Domain Class:*
  The domain class for this property is `Document` (a superclass of `ProjectInitiationDoc`). This signifies that instances of `Document` can participate in the relationship as subjects.
- *Range Class:*
  The range class is `ProjectExecutive`, indicating that instances of `ProjectExecutive` can be objects associated with the `documentCreatedBy` property.
- *Data Property:*
  The data property `documentAsActorsContribution` indicates the measure of the contribution as `xsd:String` encoded MEDIUM-value

Figure 18 displays an annotation associated with the data property `documentAsActorsContribution`.

Data properties provide additional attributes to describe entities within the ontology: The label `documentAsActorsContribution` indicates the name of the data property.

The annotation text explains the purpose of the data property displayed. The text clarifies that `documentAsActorsContribution` identifies the portion of a document created by an actor (`ProjectExecutive`).

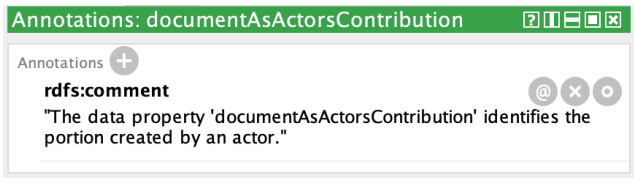


Figure 18: Data Property Annotation visualised in Protégé.

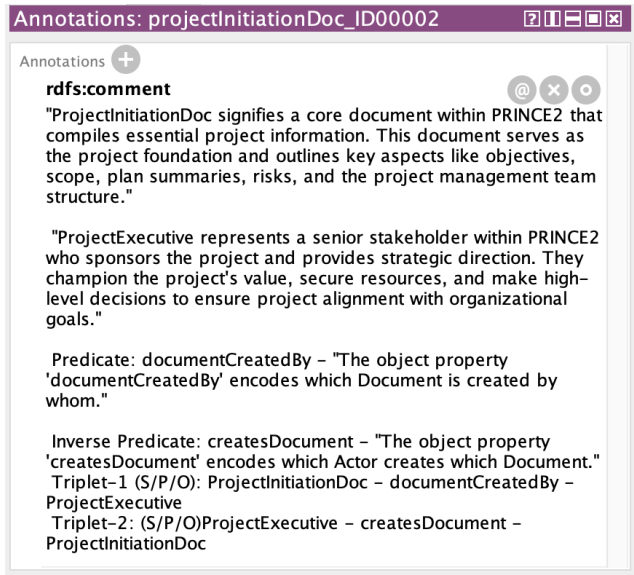


Figure 19: The annotation text clarifies the meaning of the predicate: *The object property `documentCreatedBy` encodes which Document is created by whom.* It also informs about the inverse object property.

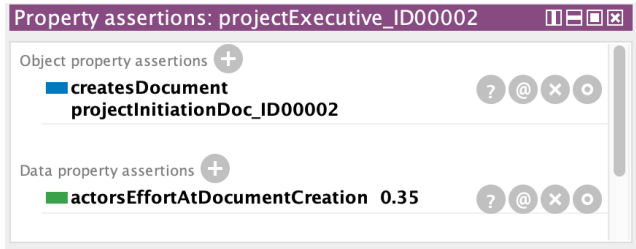


Figure 20: It displays the inverse of the original triplet (`ProjectExecutive` creates `ProjectInitiationDoc`). It also shows the data property `actorsEffortAtDocumentCreation` is associated with the object class (`ProjectExecutive`).

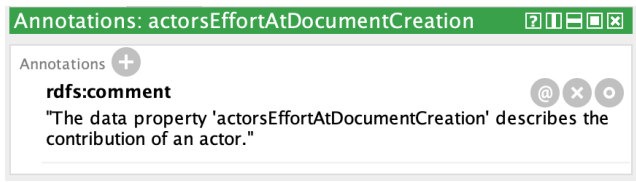


Figure 21: Which indicates a shorthand notation used within Protégé to denote that the data properties (`documentAsActorsContribution` and `actorsEffortAtDocumentCreation`) are subclasses of the predefined `owl:topDataProperty` class in the OWL hierarchy. This placement helps categorize data properties within the ontology.

Overall, these five figures (Figures 17-21) show as Protégé screenshots visually represent the information encoded within the first triplet and its inverse from the provided triplet set of Figure 15, line 925.

They illustrate the relationships between classes (`documentCreatedBy`, `createsDocument`), the data properties associated with the relationships

(`documentAsActorsContribution` and the `actorsEffortAtDocumentCreation`), as well as, how these properties connect to specific instances within the ontology.

7. Apache Jena vs OWLschemafy

The field of ontology development offers a variety of tools and frameworks to support the creation and management of knowledge structures. This chapter compares OWLschemafy, the ontology development framework discussed throughout this report, with Apache Jena, another prominent option within the Semantic Web domain.

By highlighting the strengths and considerations associated with each approach, this chapter aims to provide a more comprehensive understanding of OWLschemafy's value proposition within the landscape of ontology development tools. The comparison of Apache Jena and OWLschemafy custom framework for OWL ontology development is as follows.

7.1 Programmer- Vs. Ontology-Orientation:

- *Apache Jena:* Leans more towards programmer-oriented. It provides low-level APIs for building RDF models, manipulating statements, and interacting with OWL features. Users need to write Java code to define classes, properties, and axioms.
- *OWLschemafy:* Seems more ontology-oriented. It uses a text-file based approach with control keywords, making it potentially easier for users familiar with ontology concepts but without extensive programming experience.

7.2 Bottom-Up Vs. Top-Down Approach:

- *Apache Jena:* Offers a bottom-up approach. Users start by creating individual statements (triples) and build the ontology incrementally.
- *OWLschemafy:* Promotes a more top-down approach. Users define class disjointness and object properties with annotations and relationships upfront, providing a higher-level view.

7.3 Additional Comparative Points:

- *Scalability:* Jena is well-suited for large and complex ontologies due to its programmatic flexibility and integration with other RDF tools. OWLschemafy's scalability might be limited for very large ontologies as managing control files and parsing them could become cumbersome.
- *Interoperability:* Jena offers seamless interoperability with other RDF and OWL tools due to its adherence to W3C standards. OWLschemafy's interoperability requires Protégé, as conversion tool for use with other OWL parsers or reasoners.
- *Reasoning Support:* Apache Jena offers integration with various OWL reasoners, enabling automated reasoning and inference capabilities within ontologies. While OWLschemafy also supports reasoning, it currently relies on the Protégé ontology editor for this functionality.

8. Conclusion

8.1 The Power Of User-Centric Control:

This brief report has introduced OWLschemafy, a text-driven ontology generator that revolutionizes the approach to OWL development. By leveraging the accessibility and flexibility of text files, OWLschemafy empowers users to construct ontologies in a streamlined and user-friendly manner.

8.2 Key Features And Benefits:

- *Intuitive Text-Based Control:* OWLschemafy departs from complex ontologylanguages, employing readily understandable text files. This intuitive interface lowers the barrier to entry for ontology development, making it accessible to a wider range of users.
- *Enhanced User Control and Flexibility:* Users retain granular control over the ontology creation process through the detailed specifications within the text files. This flexibility empowers users to tailor the ontology structure and content to their specific needs.
- *Streamlined Workflow and Efficiency:* OWLschemafy automates the generation of OWL ontologies based on the user-provided control files. This automation significantly reduces the time and effort required for ontology development compared to traditional methods.
- *Seamless Integration with Protégé:* The tab-based indentation structure within the control files ensures compatibility with Protégé, a popular ontology editor. This allows users to seamlessly switch between text-based control and Protégé's visual editing environment.

8.3 Potential For Diverse Use Cases:

OWLschemafy's unique approach transcends the limitations of traditional methods, opening doors for various use cases. Researchers can leverage it to create ontologies for diverse domains, streamlining knowledge representation and fostering collaboration.

Developers can utilize OWLschemafy to generate ontologies for semantic web applications, promoting interoperability and knowledge exchange. Educators can introduce ontology development concepts to students using the intuitive text-based interface, enhancing a deeper understanding of knowledge representation.

Finally, it is important to remember that the generated OWL file is compatible with most graph databases. However, a detailed discussion of the author's SPARQL queries executed using the ontotext GraphDB software tool falls outside the scope of this description.

8.4 Looking Ahead – A Future Of Scalable And Enhanced Development:

The ongoing development of OWLschemafy explores promising avenues for future advancements. Envisioning the creation of highly expansive ontologies, the integration of relational databases as a control mechanism presents a compelling strategy for enhanced scalability. Furthermore, seamless integration with automated reasoning tools holds immense potential for ensuring the logical consistency and robustness of generated ontologies.

9. Software Background

The software development environment consists of the following components:

Operating System: macOS Monterey, (Version 12.5.1) running on an iMac (Retina 5k, 27-inch, Late 2015)

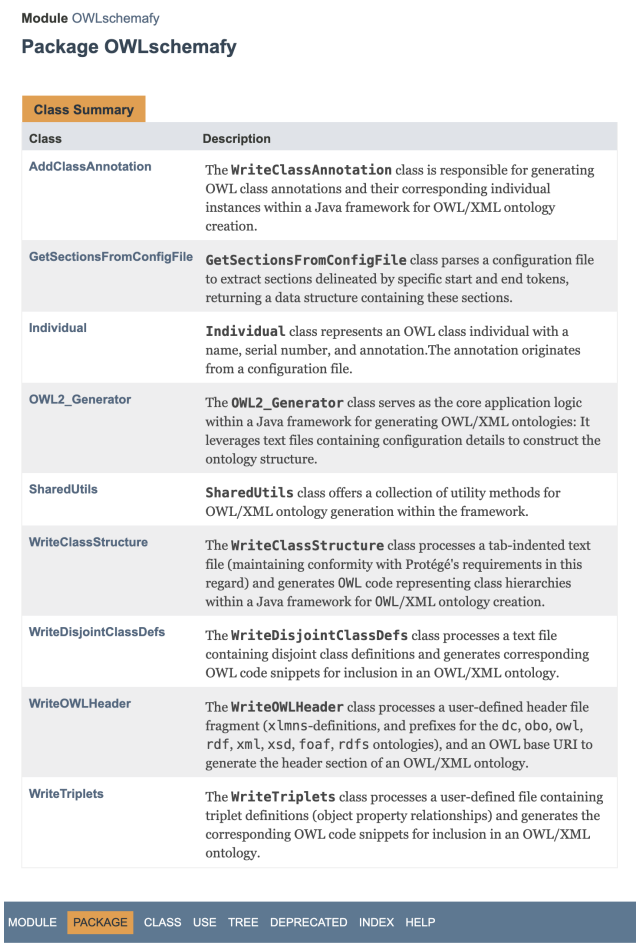
Java Runtime: Java™ SE Runtime Environment (build 13.0.2+8)

Java Virtual Machine: Java HotSpot™ 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)

Integrated Development Environment: Eclipse IDE for Java Developers (includes Incubating components), Version: 2021-12 (4.22.0), Build id: 20211202-1639

Protégé: Open-source ontology editor and framework for building intelligent systems, Version 5.6.3

Comprehensive documentation generated by JavaDoc (Figure 22) and all source files are readily available.



Module OWLschemafy

Package OWLschemafy

Class Summary

Class	Description
AddClassAnnotation	The **WriteClassAnnotation** class is responsible for generating OWL class annotations and their corresponding individual instances within a Java framework for OWL/XML ontology creation.
GetSectionsFromConfigFile	**GetSectionsFromConfigFile** class parses a configuration file to extract sections delineated by specific start and end tokens, returning a data structure containing these sections.
Individual	**Individual** class represents an OWL class individual with a name, serial number, and annotation.The annotation originates from a configuration file.
OWL2_Generator	The **OWL2_Generator** class serves as the core application logic within a Java framework for generating OWL/XML ontologies: It leverages text files containing configuration details to construct the ontology structure.
SharedUtils	**SharedUtils** class offers a collection of utility methods for OWL/XML ontology generation within the framework.
WriteClassStructure	The **WriteClassStructure** class processes a tab-indented text file (maintaining conformity with Protégé's requirements in this regard) and generates OWL code representing class hierarchies within a Java framework for OWL/XML ontology creation.
WriteDisjointClassDefs	The **WriteDisjointClassDefs** class processes a text file containing disjoint class definitions and generates corresponding OWL code snippets for inclusion in an OWL/XML ontology.
WriteOWLHeader	The **WriteOWLHeader** class processes a user-defined header file fragment (xlmns-definitions, and prefixes for the dc, obo, owl, rdf, xml, xsd, foaf, rdfs ontologies), and an OWL base URI to generate the header section of an OWL/XML ontology.
WriteTriplets	The **WriteTriplets** class processes a user-defined file containing triplet definitions (object property relationships) and generates the corresponding OWL code snippets for inclusion in an OWL/XML ontology.

MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Figure 22: OWLschemafy development documentation generated by Java Doc.

Java applications are known for their platform independence. As an illustration, Figure 23 below demonstrates the execution of OWLschemafy within a terminal window on a macOS system.

```
~/Desktop/PRINCE2_Ontology/RunableEnv % bp2
OWLschemafy V1-002 started from /Users/HlasznyE/Desktop/PRINCE2_Ontology/RunableEnv/
    asserted   10 header annotation(s)
    declared  349 class(es)
    declared  348 subclass(es)
    declared  319 disjoint class(es) in different groups
    asserted  348 class annotation(s)
    defined  1126 individual(s)
    defined   563 triplet(s)
OWLschemafy V1-002 has succesfully been completed:
Result: /Users/HlasznyE/Desktop/PRINCE2_Ontology/prince2_ontology.owl
~/Desktop/PRINCE2_Ontology/RunableEnv %
```

Figure 23: OWLschemafy Java application running on terminal window.

10. References

[1] Allemang, D., Hendler, J. A. (2012). Semantic web for the working ontologist effective modeling in RDFS and Owl. Dean Allemang, Hendler. J.A.

[2] Arp, R., Smith, B., Spear, A. D. (2015). Building ontologies with basic formal ontology. MIT Press

[3] Curé, O., Blin, G. (2015). RDF database systems: Triples Storage and SPARQL query processing. Morgan Kaufmann

[4] DuCharme, B. (2011). Learning Sparql: Querying and updating with SPARQL 1.1. O'Reilly

[5] Hofweber, T. (2018). Ontology and the ambitions of metaphysics. Oxford University Press

[6] Horridge, M. (2011). A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. The University Of Manchester Press

[7] Ontology, conceptualization and epistemology for information systems, Software Engineering and Service science Miguel-Angel Sicilia, Christian Kop, Fabio Sartori. (2010). Springer

[8] Owl. OWL - Semantic Web Standards. (n.d.) https://www.w3.org

[9] Segaran, T., Evans, C., Taylor, J. (2009). Programming in semantic web. O'Reilly

[10] Yu, L. (2014). A developer's guide to the semantic web. Springer Berlin Heidelberg

Online Content

Any control and source data, extended data, supplementary information, and author details are available electronically.

Base URL:	http://www.edithlaszny.eu/ontology/OWLschemafy/
Present paper:	paper/OWLschemafy_V_2_002.pdf
LaTeX source:	paper/OWLschemafy_V_2_002.tex
Figures:	paper/figures/
Control files:	RunableEnv/*.txt
Java sources:	eclipseWorkSpacePRINCEP2/*
Javadoc:	javadoc/index.html
Author's data:	authorsData/*.pdf
OWL Ontology:	RunableEnv/resultOntology/prince2_ontology.owl

The author declares no conflicts of interest, including any competitive interests.
This paper has been typeset from a TEX/LATEX file prepared by the author.