OLIVAW: ACIMOV's GitHub robot assisting agile collaborative ontology development

Nicolas Robert
Inria, UNICA, *CNRS, I3S*
Biot, France
0009-0009-2595-6168

Fabien Gandon
Inria, UNICA, *CNRS, I3S*
Biot, France
0000-0003-0543-1232

Maxime Lefrançois
Mines Saint-Etienne, Univ Clermont Auvergne,
INP Clermont Auvergne, CNRS, UMR 6158 LIMOS,
Saint-Étienne, France
0000-0001-9814-8991

***Abstract*—Agile and collaborative approaches to ontologies design are crucial because they contribute to making them user-driven, up-to-date, and able to evolve alongside the systems they support, hence proper continuous validation tooling is required to ensure ontologies match developers' requirements all along their development. We propose OLIVAW (Ontology Long-lived Integration Via ACIMOV Workflow), a tool supporting the ACIMOV methodology on GitHub. It relies on W3C Standards to assist the development of modular ontologies through GitHub Composite Actions, pre-commit hooks, or a command line interface. OLIVAW was tested on several ontology projects to ensure its usefulness, genericity and reusability. A template repository is available for a quick start. OLIVAW is published under the LGPL-2.1 license and archived on Software Heritage and Zenodo.**

***Index Terms*—agile, collaborative, ontology engineering, testing, GitHub, continuous integration, DevOps.**

I. Introduction

Ontologies, schemata, and vocabularies, are more and more used in information systems. They provide a structured way to define, organize, and link knowledge, enabling more effective data and application integration. They act as an underlying reference for knowledge representation, allowing information systems to achieve semantic interoperability in increasingly complex and distributed digital ecosystems.

In that context, agile and collaborative approaches to ontology development are crucial to make ontologies user-driven, consensual, up-to-date, and able to evolve alongside the systems they support. Mirroring the best practices of modern software development, ontology development benefits from being iterative, responsive to feedback, and collaborative, ultimately resulting in ontologies that are more resilient, useful, and aligned with user needs [1].

The distributed version management software Git is widely adopted in collaborative software development. Git platforms such as GitHub or GitLab have popularised development workflows centred on branches, forks, and pull/merge requests, support agile project management with issues and milestones, and support DevOps principles through continuous integration and deployment. These continuous integration mechanisms are ensured by *GitHub Actions* on Github and *GitLab CI/CD* on GitLab. GitHub also provides the *GitHub Gist* service allowing to share text snippets referred to as *gist files*.

Leveraging agility and DevOps principles with a standard git-based approach that is relying on the CI/CD mechanisms is a distinguishing feature of the ACIMOV ontology engineering methodology (Agile and Continuous Integration for Modular Ontologies and Vocabularies) [1]. For example, the issue tracker can be used to discuss and document ontology engineering decisions. Labels and milestones can be used to categorise and prioritise requirements and manage the backlog. Continuous integration and deployment workflows can perform syntactic and semantic checks on the ontology, and generate and publish its documentation.

This paper presents OLIVAW (Ontology Long-lived Integration Via ACIMOV Workflow), a tool that supports the ACIMOV methodology on GitHub to assist agile collaborative ontology development. OLIVAW relies on W3C Standards, and is usable as a standalone library through a command line interface, as pre-commit hooks, or as GitHub Actions. OLIVAW focuses on continuous integration and can be combined with other existing software for other aspects of development such as documentation or deployment. OLIVAW is *FAIR* since its (1) *Findable* by its Zenodo DOI, Software Heritage SWHID and HAL-ID; (2) *Accessible* by Zenodo, Software Heritage, GitHub, Pipy ; (3) *Interoperable* by its use of development standards such as GitHub Actions, PyPI, W3C standards ; and (4) *Reusable* by its LGPL license, its documentation and repository template.

The next sections describe OLIVAW starting with the related work (Section II) and terminology (Section III). We then introduce the general principles of OLIVAW and each family of tests it can perform and reports it produces (Section IV). We explain the different alternatives to deploy OLIVAW (Section V) and report on our experimentations and evaluations (Section VI), before concluding.

II. Related Work

Several ontology engineering methodologies are inspired by the principles of agile software engineering, which promote collaboration between developers and stakeholders by producing regular updates of the product. Among these methods, AMOD [2] and CD-OAM [3] are based on SCRUM. XPOD [4], eXtreme ontology method [5], and eXtreme Design (XD) [6] are based on eXtreme Programming. The Lean Ontology Development (LOD) [7] is inspired by the Lean approach: Build-Measure-Learn. SAMOD [8] is revisiting the motivating scenarios and competency questions of Uschold

and Gruninger [9], additionally considering ontology modules and test-driven development. Some tools exist to support some of these methodologies. For example, XDTesting [10] is an automated test tool supporting the XD methodology. Three types of tests are available, namely the Competency Question Verification which consists in testing a competency question over a dataset, an Inference Verification which consists in checking if the entailment creates some required triples, and the Error Provocation which consists in feeding a modelet with a dataset that is incomplete or incorrect in order to check how the entailment behaves.

Just as agility aims to improve collaborations between software project customers and developers, DevOps improves collaborations between developers and IT operations professionals. Jenkins, Travis CI, Circle CI, GitLab CI/CD, Github Actions, are all frameworks that allow to specify CI/CD workflows that will be executed automatically when, for example, a commit is pushed to the server. Before even these frameworks existed, VoCol [11] and OnToology [12] supported CI/CD in ontology engineering using Github applications[1]. The *Ontology Development Kit* (ODK) [13] uses Travis CI to run CI/CD workflows with the ROBOT tool [14] developed by the Open Biological and Biomedical Ontologies (OBO) community. Different ontology projects use CI/CD workflows, such as the Financial Industry Business Ontology (FIBO) [15], the International Data Spaces Information Model (IDSA) [16], the CASE Cyber Ontology[2] and the Smart Applications REFerence Ontology (SAREF) [17]. The latter uses a dedicated development methodology and a dedicated software, the SAREF-Pipeline [18], that is used to verify the quality of contributions and generate the documentation website. To set up CI/CD workflows for new ontology projects, some Github actions are available on the Github marketplace for running RDFLint[3], validating RDF syntaxes[4,5], or validating RDF files against SHACL shapes[6] or ShEx [19].

ACIMOV [1] extends SAMOD and explicitly adopts the standard git-based approach for coding, leveraging agility and DevOps principles. It addresses modularity needs with ontologies being designed as a set of stand-alone modules called *modelets*, describing specific aspects and covering focused sets of requirements [1]. The assignment and the backlog of modelets is done through the issue tracker. ACIMOV emphasizes collaboration among team members, stakeholders, and domain experts throughout the ontology development process. Each ontology engineer contributing to the repository can take the charge of developing and testing a modelet's artifacts, typically including: motivating scenarios, competency questions, specification of classes, properties and individuals, some RDF test data for a use case, SPARQL queries representing competency questions to be tested over test data, etc. Following ACIMOV,

[1]https://docs.github.com/en/developers/apps
[2]https://github.com/marketplace/actions/case-ontology-validator
[3]https://github.com/marketplace/actions/setup-rdflint
[4]https://github.com/marketplace/actions/rdf-syntax-check
[5]https://github.com/marketplace/actions/validate-rdf-with-jena
[6]https://github.com/marketplace/actions/validate-shacl

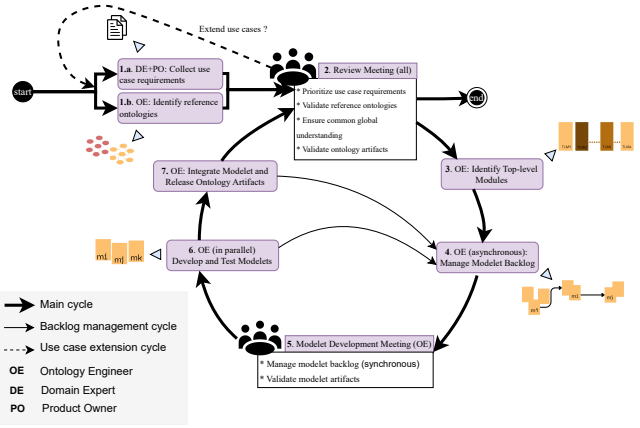


Fig. 1: Overview of the ACIMOV methodology [1]

modelets require *testing* to check the quality and coherence of the ontological description, *validation* of successful testing results and a decision from the ontology engineers for integration to a new version of the ontology and a release. In other words, there is a need for tooling. ACIMOV is pure methodology with no tooling and the OLIVAW framework introduced in this article is meant to provide such tooling, supporting continuous integration for ACIMOV. It covers a range of tests including Competency Question Verification and Inference Verification, and provides even more e.g. ontology profile diagnoses.

In the next section we recall some RDF vocabularies and core concepts from SAMOD and ACIMOV that are needed to understand OLIVAW.

III. Terminology

OLIVAW is grounded in the Semantic Web stack using RDF [20], OWL [21], SPARQL [22] and SHACL [23]. It also uses PROV-O [24], and the Evaluation and Report Language vocabulary (EARL) [25].

EARL is an RDF vocabulary allowing the representation of test reports. Each test report entry is seen as **Assertion**, which is an aggregation of : an **Assertor** (the actor who ran the test), a **TestSubject** (the tested resource, which can be composed of several files) a **TestCriterion** (the constraint to be tested against the TestSubject) and a **TestResult** (the result of the test that was run, which is an aggregation of Outcomes that provide precise information). A TestResult is composed of outcomes that can have for output type: **Pass** if the subject passed the test; **NotTested** if the test could not be run because some prerequisite statements were not validated (example: for a car, test "gear shift" could not be run because test "turn on" was not validated); **CannotTell** if the automated test detected some unexpected behavior where a human is required to conclude if this is an error; and **Fail** if the subject failed the test.

SAMOD grounds the development of ontologies on **Motivating Scenarios**, which are practical descriptions of situations related to a domain. A motivating scenario is used to justify the introduction of new terms in some iteration of the ontology development. **Modelets** are stand-alone models that for-

malize the new terms of a motivating scenario. Requirements are first expressed informally using **Competency Question** in natural language, then are implemented in SPARQL or any other formal language. **Dataset** (ABox in SAMOD) are small data graphs that model part of the motivating scenario, and are used to validate modelets against formalized competency questions. The final ontology is the union of all modelets. ACIMOV extends SAMOD to support the development of modular and multi-domain ontologies. Motivating scenarios are grouped in domains. Modules are ontology fragments that merge an arbitrary set of modelets. **Use-cases** are larger data graphs that illustrate all of the motivating scenarios related to a specific domain. As OLIVAW supports ACIMOV, three types of tests will be involved depending on the type of resources they target: (1) **Model tests** for modelets and modules, which we refer to as **ontology fragments**; **Data tests** for datasets and use-cases, which we refer to as **data fragments**; and **Query tests** for formal competency questions.

IV. OLIVAW Framework and Dev. Workflow

OLIVAW is an open source python framework that supports ACIMOV, published under the LGPL-2.1 license. It is available as a GitHub repository[7], a PyPI package[8], a Software Heritage Archive[9] and a Zenodo record[10]. It has a full online documentation with examples[11]. It is documented using Sphinx docstrings, function signatures using `typing` and many `README.md` files for optimal code readability, maintainability and extensibility. The documentation also provides OLIVAW reports examples for several ontologies.[12]

A. OLIVAW ontology project structure

OLIVAW can support any ontology project matching the minimal folder structure of Figure 2. The OLIVAW repository template[13] offers a quick start for new ontology projects. The `src/` folder contains the ontology modules and `domains/` contains a sub-folder for each domain. Each domain folder contains a subfolder for each motivating scenario. Each motivating scenario[14] contains a modelet, a dataset and related competency questions implementations. The `use-cases/` folder contains a sub-folder for each use case. Each use case may contain one or more data fragments. The `.acimov/` folder contains files related to the ACIMOV methodology, support tools, and a `parameters.json` file setting the specific parameters of the project (see Section V). Files generated by OLIVAW go in `output/`. Finally, `custom-tests/` folder is meant to store custom tests to be run against the project (see Section IV-F).

[7]https://github.com/Wimmics/olivaw

[8]https://pypi.org/project/olivaw/

[9]https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/Wimmics/olivaw

[10]https://doi.org/10.5281/zenodo.14288084

[11]https://github.com/Wimmics/olivaw/tree/main/docs

[12]https://github.com/Wimmics/olivaw/tree/main/docs/examples

[13]https://github.com/Wimmics/Olivaw-Template

[14]Example: https://github.com/HyperAgents/hmas/tree/main/domains/logistics/create-organization

B. OLIVAW General Overview

OLIVAW builds on CORESE (Conceptual Resource Search Engine) [26], which is an open-source semantic web framework[15] implementing RDF, SPARQL, SHACL, RDFS and OWL RL. As shown on Figure 3, OLIVAW can execute tests at three different and complementary moments: (i) on the computer of the ontology developer, using the command-line actions for some project analysis. (ii) as a pre-commit hook, that will check all the files staged for commit and prevent blocking errors. (iii) As a job configured to run during a GitHub Actions CI/CD workflow, for example to analyze the project and generate reports after a push to some branch, or the creation of a pull request. Each mechanism can be installed and configured independently. OLIVAW covers Model tests (see Section IV-C), Data tests (see Section IV-D), and Query tests (see Section IV-E). Custom tests are explained in Section IV-F. Tests generate reports in Turtle (see Section IV-G) and Markdown (see Section IV-H). These reports are saved in the `.acimov/output/` folder, on the local drive for tests run locally or on the server for tests run by GitHub Actions, supporting actual branch state control and tracking over time.

During a GitHub Action, dynamic badges are updated on the branch README.md file to provide some basic metrics about how many outcomes are to be found on this project for each outcome type (as defined in Section III), and also the ontology (merged modules) compatibility w.r.t. each OWL profile (EL/QL/RL). Each of these badges is clickable and points to its related report. OLIVAW can follow specific settings to adapt to some particular project needs, such as errors to consider as blocking or tests to discard, through a project file `.acimov/parameters.json` (see Section V).

The OLIVAW ontology[16] instanciates some EARL TestCriterion and extends the EARL Fail class to capture the error severity (`MinorFail` and `MajorFail` classes). Any error is considered as `MinorFail` unless it appears in the project list of blocking errors to be considered as a `MajorFail`. This ontology also includes some terms used in the generated reports e.g. `VersionedEntity`, is a class of entities used in the OLIVAW tests, typing the different versions of an entity.

C. Model tests

Model tests cover different levels of checking, from low-level syntax validation to matching model expectations and best practices. These tests are applied on: (1) each module[17] individually, (2) each modelet[18] individually, (3) each module merged with related terms from a given modelet, (4) the merge of all the syntactically correct modules and (5) all the syntactically correct modules and modelets merged together (see Section III for terminology). All sources of errors are covered on the module level, at the integration stage when

[15]https://github.com/Wimmics/corese/

[16]https://ns.inria.fr/olivaw/

[17]Module example from hMAS: https://github.com/HyperAgents/hmas/blob/main/src/regulation.ttl

[18]Modelet example from hMAS: https://github.com/HyperAgents/hmas/blob/main/domains/manufacturing-environments/discover-signifiers/onto.ttl

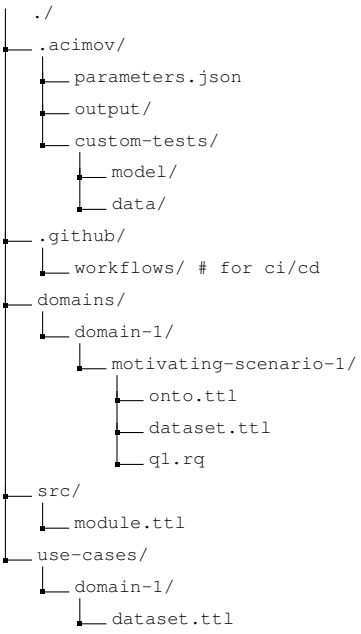


Fig. 2: Repository structure

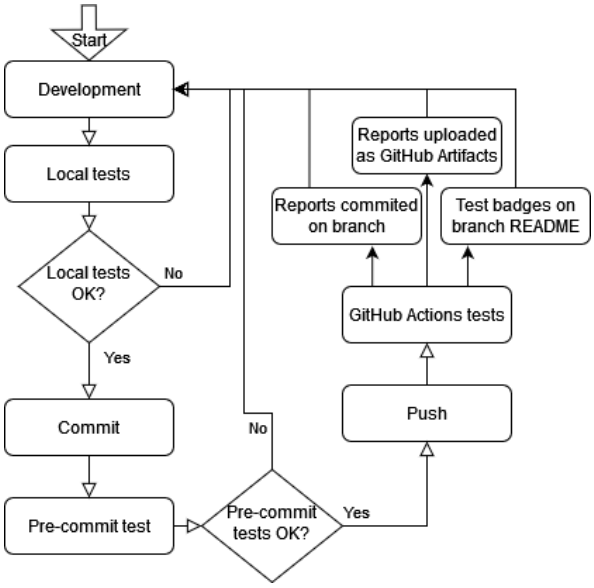


Fig. 3: Three complementary executions of tests by OLIVAW

merging the terms of a modelet in a module or at the whole ontology level by merging all the ontology modules and modelets together. Additional custom tests can be added to the model tests for a specific project (see Section IV-F).

OLIVAW includes the following tests by default. **RDF syntax validity**: the test subject is loaded in CORESE and should not return any syntax error. **Terms are linked to their ontology**: this test consists in checking if for each ontology term, there exists a `rdfs:isDefinedBy` property with a value for this term. **Domain or range out of vocabulary**: this test consists in checking if each triple with a subject namespaced in the vocabulary and predicate `rdfs:domain` or `rdfs:range` has an object namespaced in the ontology or points to an external URI. **Proper use of subset properties**: this test checks if any `rdfs:subClassOf` predicate has a subject or object that is of type `rdf:Property` and if any `rdfs:subPropertyOf` predicate has a subject or object that is of type `rdfs:Class`. **Differentiation of terms**: checks if any pair of terms has a Levenshtein distance under a given threshold defined in the configuration file (see Section V), for instance properties `hasName` and `has_name`. **English labels are provided**: this test checks if all the ontology terms have an `rdfs:label` with a literal value tagged as English (`@en`). **OWL 2 RL consistency**: this checks the logical consistency of the model within that profile, but not the compatibility with that profile.[19] **OWL profile compatibility**: this checks if the tested resource is compatible with the OWL profiles *RL*, *QL* and *EL*.

[19]e.g. an ontology containing a `owl:disjointUnionOf` could be found consistent by an OWL 2 RL reasoner but would not be compatible with that OWL profile

D. Data tests

Data tests check if the data fragments using the vocabulary respect some standards. These tests are applied on each dataset[20] individually and in each use-case[21] fragment individually (see Section III). These tests include by default the following tests. **Syntax validity**, equivalent to test in Section IV-C. **OWL RL consistency test**, equivalent to the test in Section IV-C. **Known Terms**: The test checks if any test subject term belonging to the ontology namespace is not defined in the ontology. **Namespace test**: The test checks if any test subject namespace has a Levenshtein distance equal to 1 or 2 from a namespace found on prefix.cc, or from another project namespace. This allows the detection of potential typos such as the use of http/https or trailing hash/slash, and is reported as `CannotTell` since they could be correct yet unknown to OLIVAW.

E. Query tests

Query tests check if the competency questions are matching the expectations and best practices. These tests include: **Syntax tests** during which the construction of a SPARQL query AST (abstract syntax tree) is attempted by CORESE, checking the validity of the query syntax. **Query type test** : any query should use either the `SELECT` or `ASK` clause to match an open or closed question in natural language. **URI validity test**: any URI in the query should conform to the RFC 3986 [27]. **Namespace test**: equivalent to Section IV-D.

[20]Dataset example from hMAS: https://github.com/HyperAgents/hmas/blob/main/domains/manufacturing-environments/discover-organization/dataset.ttl

[21]Use-case example from hMAS: https://github.com/HyperAgents/hmas/blob/182-uc-manufacturing/use-cases/manufacturing/cup-production.ttl

F. Extension points and Custom tests

Any ontology development project may have additional requirements specific to an application, a community, a domain, an ontology management platform, a methodology, an organization, etc. Any project can integrate OLIVAW even if not following the ACIMOV architecture. It is possible to configure OLIVAW to fit a non-ACIMOV architectured project (see OLIVAW parameters documentation for more details). Moreover, SKOS(-XL) thesauri or other vocabularies may require specific tests that OLIVAW does not cover by default but can accommodate. In order to meet these requirements, OLIVAW supports the implementation of custom tests that can be added to model tests or data tests.

These tests can be implemented as graphs of SHACL shapes that will be tested against the fragments. This feature is meant to enable custom model or data test implementation before ontology or data development, which is a commonly adopted principle in software development (e.g. in V model methodology or in Test Driven Development). This feature also allows custom test reusability from one project to another without any change required on the test itself.

However SHACL can lack the required expressivity for some tests. For example, implementing the detection of a cycle in the hierarchy of properties cannot be done in standard SHACL. Moreover the basic use case of SHACL is to validate a dataset against shapes, targeting some nodes based on classes, URIs or properties that are already known in advance. However none of these targets are known in advance in an ontology development context, which complicates the proper node targeting for upstream test implementation.

In order to make these custom tests powerful and convenient, it is possible to take advantage of all the non-standard SHACL features in CORESE. When additional expressivity is needed, CORESE's implementation of SHACL-SPARQL[22] can be used. It brings in SHACL all the expressivity available in SPARQL. CORESE also allows to target triples instead of nodes and brings many new options to implement powerful SHACL paths in order to target and find the fine-grain patterns that are expected to be detected in any specific ontology development project. This feature is fully documented and several examples can be found in OLIVAW custom tests documentation and examples[23].

G. Turtle report format

The first report format generated by OLIVAW is a RDF format in Turtle. It is meant to be processable and queried by machines. EARL is the vocabulary used to express the turtle reports. Two extensions have been made compared to the EARL ontology developer guide[24].

The Fail class mentioned in Section IV-B has been extended into several subclasses in order to express some levels of severity for a given Fail outcome, namely `olivaw:MajorFail` for an outcome that would block the ontology to be deployed to production, and `olivaw:MinorFail` otherwise.

The `earl:Assertor` involved in the different assertions of the test is described as an activity that involves: a person associated as developer; the ontology used as tested subject; the executed OLIVAW test script used as test suite; both reports files (Turtle and Markdown) as activity generations. This activity is described using the PROV ontology as shown in Figure 4a. In this format, generated files are named by their GitHub file URIs if GitHub Actions ran the test and by their local file paths if tests were run locally. In order to qualify associations and usages, several `prov:Role` instances are defined in the OLIVAW ontology. A property has also been defined to qualify the activity generations. It is used in the report as shows Figure 4b.

The test activity used two `prov:Entity` instances, namely the tested project and the test suite. These entities are of type `olivaw:VersionedEntity`. Resources of this subclass of `prov:Entity` are identified (with an `owl:hasKey` axiom) by the host URL and the entity version. When a test is run locally, it may apply to a local repository state that has no commit hash yet. In that case, the tested project version is a hash of files hashes in order to identify it from another different uncommited local state. Figure 4c shows this implementation.

For each test, the `earl:TestSubject` is composed of URIs pointing to the files that were involved in the Assertion, in the origin server, as illustrated in Figure 4d. Finally, the result is a node that can point to one to several outcomes for each Assertion. Each assertion in the report is an `earl:Assertion`. This assertion implementation can be seen in Figure 4e.

H. Markdown report format

The markdown report format is derived from the turtle report to provide a human-friendly version. Since this report can be very long, it includes hyperlinks to be browsed, searched and navigated easily on GitHub and to be fully integrated in that environment. The report is composed of several sections that we review next. Some examples of reports for model tests are available online.[25]

The first section is a short explanation about the report itself and how it was obtained from the Turtle version. The second section provides information about the assertor, including the developer name, how the report generation was triggered, the triggered test suite and finally the date time of the tests. When a local test is run, the project version is identified by the hash of all the project file hashes that are not ignored. This allows to distinguish the head commit version from an uncommitted local version or different uncommitted local versions. In any local test report, the hash corresponding to the commit that uncommitted version is derived from will also be specified.

The third section, showed in Figure 5, is a statistic summary providing the total number of entries in the report and the

[22]https://www.w3.org/TR/shacl/#shacl-sparql

[23]https://github.com/Wimmics/olivaw/blob/main/docs/custom-tests.md

[24]https://www.w3.org/WAI/ER/EARL10/WD-EARL10-Guide-20120125

[25]https://github.com/Wimmics/olivaw/blob/main/docs/examples/hmas/model-test-manual-NicoRobertIn-2024-12-19T08-59-52.md

```
_:assertor a earl:Assertor, prov:Activity ;
    dcterms:title "......." ; dcterms:description "......" ;
    prov:endedAtTime "2024-09-25T15:21:22.513553"^^xsd:dateTime ;
    prov:generated <file:///....md>, <file:///....ttl> ;
    prov:qualifiedAssociation [
        a prov:Association ; prov:hadRole olivaw:tester ;
        prov:agent _:tester ;
    ] ;
    prov:qualifiedUsage [
        a prov:Usage ; prov:hadRole olivaw:test_suite ;
        prov:entity <https://.../model/suite.py>
    ], [
        a prov:Usage ; prov:hadRole olivaw:tested_project ;
        prov:entity _:intermediateSnapshot
    ] ;
    prov:used _:snapshot, <https://.../model/suite.py> ;
    prov:wasAssociatedWith _:tester .

_:snapshot a olivaw:VersionedEntity ;
    dcterms:date "2024-09-25T15:21:11.561240"^^xsd:dateTime ;
    dcterms:hasVersion "5a63633e10a78f4bac71749caf63b057f5601daa" ;
    olivaw:hostedAt <https://github.com/HyperAgents/hmas> ;
    olivaw:isOnBranch "main" ;
    olivaw:patchedBy _:tester ;
    olivaw:patchedFrom "3b0b11eccd899dd65ac9c3bbf0c043e28b61b46b" .

<https://.../v0.0.6/.../model/suite.py> a olivaw:VersionedEntity ;
    dcterms:hasVersion "v0.0.6" ;
    olivaw:hostedAt <https://.../model/suite.py> .
```

```
<file:///....md> a prov:Entity ;
    prov:generatedAtTime "2024-09-25T15:21:22.513553"^^xsd:dateTime ;
    prov:qualifiedGeneration [ a prov:Generation ;
            prov:activity _:assertor ;
            olivaw:generatedAs olivaw:markdown_report ] .

<file:///....ttl> a prov:Entity ;
    prov:generatedAtTime "2024-09-25T15:21:22.513553"^^xsd:dateTime ;
    prov:qualifiedGeneration [ a prov:Generation ;
            prov:activity _:assertor ;
            olivaw:generatedAs olivaw:turtle_report ] .

_:module-src-regulation a earl:TestSubject ;
    dcterms:hasPart <https://github.com/HyperAgents/.../regulation.ttl> ;
    dcterms:identifier "module-src-regulation" ;
    dcterms:title "Standalone module src/regulation.ttl from branch main"@en .

[] a earl:Assertion ;
    earl:assertedBy _:assertor ;
    earl:result [ a earl:TestResult ;
        earl:outcome [ a earl:Pass ;
            dcterms:description "Each rdfs:range is defined within the fragment"@en ;
            dcterms:identifier "range-out-of-vocabulary" ;
            dcterms:title "Ranges properly defined"@en ] ] ;
    earl:subject _:modelet-logistics-coordinate-activities ;
    earl:test olivaw:domain-and-range-referencing .
```

Fig. 4: Turtle reports extracts

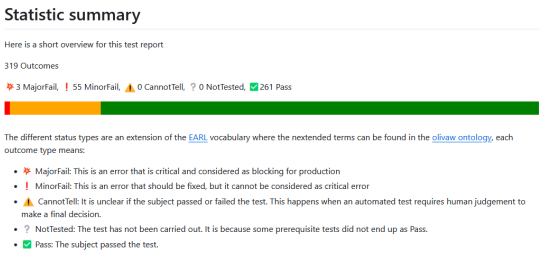


Fig. 5: Statistics summary of a markdown report

number of outcomes for each type of severity an outcome can have, with a very simple bar chart and a brief explanation of the meaning of each severity.

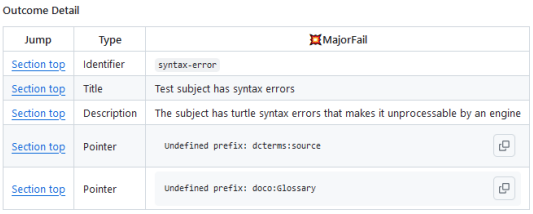
The next sections are providing details about the outcomes for each type of outcome present in the report. Examples are provides in Section VI. Each of them begins with a table that is a summary of the outcome containing the subject, the criterion that is related to that outcome, the title of the outcome and a link that leads directly to the detailed information about that given outcome. Each of these sections ends with an aggregation of the outcome details with, for each outcome, tables providing subject, criterion and outcome information. The subject table provides the subject natural language description and the list of the files composing the subject with their links to their GitHub related pages. The criterion table provides the criterion title and description. The outcome table provides the outcome title, description and a list of pointers. These pointers can be URIs, snippets of code or natural language messages providing useful information about the outcome. Each of these detailed outcome has a link bringing back to the related outcome summary line so the user can browse the summary, check the details of an outcome, and resume browsing again.

V. Deployment of OLIVAW

OLIVAW can be used: (i) locally as a python package providing a command line tool that can be run at development time; (ii) locally as a pre-commit connector for automatic test before commit and (iii) remotely as GitHub Composite Actions triggering on events such as push or merge for continuous validation. It can be configured as described in the rest of this section.

As a command line tool, OLIVAW also provides repository management commands such as for: (i) initializing a repository or a new branch; (ii) testing the project from a model, data or query perspective and (iii) flushing the reports folder. The OLIVAW command line documentation is available online[26]. OLIVAW has a pre-commit connector usable from any repository. On trigger, it analyzes each file staged for commit and blocks commit in case it detects a blocking error. If the commit is blocked, the console outputs useful information

[26]https://github.com/Wimmics/olivaw/blob/main/docs/commands.md



Outcome Detail

Jump	Type	MajorFail
Section top	Identifier	`syntax-error`
Section top	Title	Test subject has syntax errors
Section top	Description	The subject has turtle syntax errors that makes it unprocessable by an engine
Section top	Pointer	`Undefined prefix: dcterms:source`
Section top	Pointer	`Undefined prefix: doco:Glossary`

Fig. 6: Example of model syntax error in markdown

about the files and the related blocking errors. This connector is agnostic to the way git is used (command line, desktop or anything else). The OLIVAW pre-commit hook documentation is available online[27].

GitHub Actions is a continuous integration mechanism on GitHub allowing script execution on GitHub side in reaction to some repository event. A *GitHub Composite Actions* is a ready-to-use *GitHub Action*s script callable from a *GitHub Actions* script of another repository. OLIVAW has two *GitHub Composite Actions* in order to react to some actions that are made on the repository. The first *GitHub Actions* concerns the tests. It can be triggered each time a change is pushed to the repository and launches each of the desired tests. These tests can then : (a) be uploaded as GitHub artifacts in order to keep track of the history of the state of a branch; (b) be committed on the branch in order to have on GitHub the actual status of the branch and (c) update the gist files that are used to dynamically display the proper data in the `README.md` file badges. The second *GitHub Actions* assists branch creation and updates the branch badges links to the readme file when a branch is created. It also creates new gist files on GitHub to have independant badges for each branch. The OLIVAW Composite Actions documentation[28] is available online.

An ontology development project can have specific requirements, including which errors are considered as blocking, which tests from the suites should not run, which files should not be tested, or a combination of these factors (e.g., which test should not run on a given file). This may be configured in the `parameters.json` file (see the OLIVAW parameters file documentation online[29]).

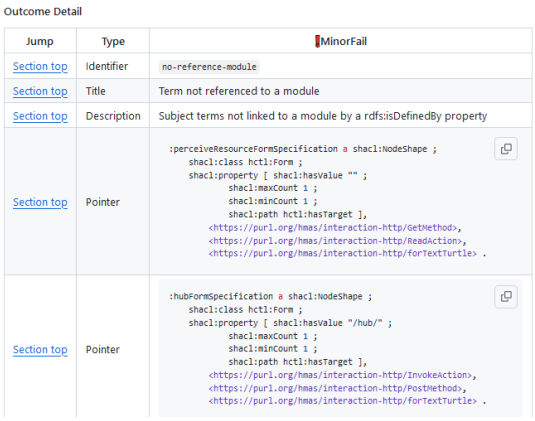
VI. Experimentation and Evaluation

A. First use case: the hMAS ontology project

This section illustrates the use of OLIVAW during the hMAS ontology development[30] with some examples of reports for the different types of errors we covered in the previous sections. The first example is the syntax error for which an example of report is shown in Figure 6.

Another error considered in hMAS is the lack of `rdfs:isDefinedBy` property on a ontology term, for which an example of report is shown in Figure 7.

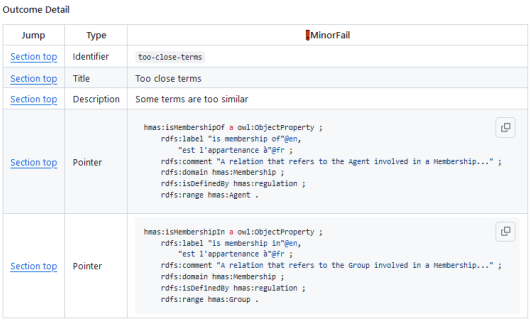
As a last example we can highlight the excessive similarity between different terms. In hMAS the pair of terms



Outcome Detail

Jump	Type	MinorFail
Section top	Identifier	`no-reference-module`
Section top	Title	Term not referenced to a module
Section top	Description	Subject terms not linked to a module by a rdfs:isDefinedBy property
Section top	Pointer	`:perceiveResourceFormSpecification a shacl:NodeShape ; shacl:class hctl:Form ; shacl:property [shacl:hasValue "" ; shacl:maxCount 1 ; shacl:minCount 1 ; shacl:path hctl:hasTarget], <https://purl.org/hmas/interaction-http/GetMethod>, <https://purl.org/hmas/interaction-http/ReadAction>, <https://purl.org/hmas/interaction-http/forTextturtle> .`
Section top	Pointer	`:hubFormSpecification a shacl:NodeShape ; shacl:class hctl:Form ; shacl:property [shacl:hasValue "/hub/" ; shacl:maxCount 1 ; shacl:minCount 1 ; shacl:path hctl:hasTarget], <https://purl.org/hmas/interaction-http/InvokeAction>, <https://purl.org/hmas/interaction-http/PostMethod>, <https://purl.org/hmas/interaction-http/forTextturtle> .`

Fig. 7: Example of term referencing error in markdown



Outcome Detail

Jump	Type	MinorFail
Section top	Identifier	`too-close-terms`
Section top	Title	Too close terms
Section top	Description	Some terms are too similar
Section top	Pointer	`hmas:isMembershipOf a owl:ObjectProperty ; rdfs:label "is membership of"@en, "est l'appartenance à"@fr ; rdfs:comment "A relation that refers to the Agent involved in a Membership..." ; rdfs:domain hmas:Membership ; rdfs:isDefinedBy hmas:regulation ; rdfs:range hmas:Agent .`
Section top	Pointer	`hmas:isMembershipIn a owl:ObjectProperty ; rdfs:label "is membership in"@en, "est l'appartenance à"@fr ; rdfs:comment "A relation that refers to the Group involved in a Membership..." ; rdfs:domain hmas:Membership ; rdfs:isDefinedBy hmas:regulation ; rdfs:range hmas:Group .`

Fig. 8: Example of term Differentiation error in markdown

`isMembershipOf` and `isMembershipIn` has a Levenshtein distance only equal to 2. The difference between these terms is clearly defined, but it could lead the users downstream to confusions and typos. Figure 8 shows how such an error is reported by OLIVAW.

B. Generalizing to different ontologies

In order to ensure the framework genericity and its usability over a large panel of ontologies, OLIVAW has also been tested against COSWOT[31] and AEC3PO[32] projects, which match the ACIMOV architecture, and against a few already deployed INRIA ontologies, namely MUNC[33], NRV[34] and PWKSO[35]. The reports mentioned here can be found in the examples section of OLIVAW repository[36].

Running the model test on COSWOT revealed a syntax error on the main branch. In file `core/coswot-saref.ttl` on line 112, a semi-column was forgotten, causing a parsing error on line 113. On top of this only blocking error, many outcomes related to profile diagnose have revealed that COSWOT and AEC3PO ontology are included neither in OWL EL, QL or RL profiles. The enumeration of the OWL RL incompatibility outcomes indicated that, in order to be OWL RL compatible, the

[27] https://github.com/Wimmics/olivaw/blob/main/docs/pre-commit.md
[28] https://github.com/Wimmics/olivaw/blob/main/docs/actions.md
[29] https://github.com/Wimmics/olivaw/blob/main/docs/parameters.md
[30] https://github.com/HyperAgents/hmas/
[31] https://gitlab.com/coswot/coswot-acimov/
[32] https://github.com/Accord-Project/aec3po
[33] https://ns.inria.fr/munc
[34] https://ns.inria.fr/nrv
[35] https://ns.inria.fr/pwkso
[36] https://github.com/Wimmics/olivaw/tree/query/docs/examples

ontologies should avoid the use of `owl:disjointUnionOf` and avoid the use of `owl:Restriction` as objects of the predicates `rdfs:subClassOf`, `rdfs:domain` and `rdfs:range`. If these changes are made on the ontology, it would make them OWL RL compatible.

The different INRIA ontologies tested are also really close to comply with the OWL RL profile. MUNC should avoid using `owl:ReflexiveProperty`. NRV is using `owl:unionOf` but not as a subclass expression and uses `owl:disjointUnionOf`. PWKSO uses restrictions as objects of `rdfs:subClassOf`. Once these incompatibility issues are highlighted by OLIVAW, it is then up to the developers to decide if these axioms are needed for the description of the domain, or if the removal of these axioms is a fair price to pay for being in a simpler OWL profile.

VII. Conclusion

In this article, we presented OLIVAW (Ontology Long-lived Integration Via ACIMOV Workflow), a tool that implements the ACIMOV methodology on GitHub relying on W3C Standards to assist the lifecycle of modularised and multi-domain ontologies. We introduced the general principles and each family of tests it can perform and the reports it produces. We explained different alternatives to use OLIVAW and we reported on our experimentation and evaluations against different ontologies of various sizes and application domains. OLIVAW is available on GitHub under the LGPL-2.1 license, with its documentation and examples in OLIVAW GitHub repository[37].

There are many other features that we are considering for OLIVAW. First to create GitLab CI/CD implementations of the existing GitHub Actions in order for OLIVAW to be fully compatible with GitLab. Secondly, to add some ontology integration control features in order to address large-scale projects. And finally to provide some deployment features and automatic tools that would help developers at other development steps than the testing step.

References

[1] F.-Z. Hannou, V. Charpenay, M. Lefrançois, C. Roussey, A. Zimmermann, and F. Gandon, "The ACIMOV Methodology: Agile and Continuous Integration for Modular Ontologies and Vocabularies," in *MK 2023 - 2nd Workshop on Modular Knowledge associated with FOIS 2023 - the 13th International Conference on Formal Ontology in Information Systems*, Sherbrooke, Canada, Jul. 2023. [Online]. Available: https://laas.hal.science/hal-04187236

[2] A. S. Abdelghany, N. R. Darwish, and H. A. Hefni, "An agile methodology for ontology development," *International Journal of Intelligent Engineering and Systems*, vol. 12, no. 2, pp. 170–181, 2019.

[3] A. Takhom, S. Usanavasin, T. Supnithi, and P. Boonkwan, "A collaborative framework supporting ontology development based on agile and scrum model," *IEICE TRANSACTIONS on Information and Systems*, vol. 103, no. 12, pp. 2568–2577, 2020.

[4] A. A. Sharifloo and M. Shamsfard, "Using agility in ontology construction." in *Formal Ontologies Meet Industry*, ser. Frontiers in Artificial Intelligence and Applications, vol. 174. IOS Press, 2008, pp. 109–119.

[5] M. Hristozova and L. Sterling, "An extreme method for developing lightweight ontologies," in *In Workshop on Ontologies in Agent Systems, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*. Citeseer, 2002.

[6] R. de Almeida Falbo, M. P. Barcellos, J. C. Nardi, and G. Guizzardi, "Organizing ontology design patterns as ontology pattern languages," in *The Semantic Web: Semantics and Big Data: 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings 10*. Springer, 2013, pp. 61–75.

[7] J. Cummings and D. Stacey, "Lean ontology development: An ontology development paradigm based on continuous innovation." in *Knowledge Engineering and Ontology Development*, 2018, pp. 365–372.

[8] S. Peroni, "Samod: an agile methodology for the development of ontologies," in *OWL: Experiences and Directions Workshop, OWL reasoner evaluation workshop, OWLED-ORE 2016*, 2016, pp. 1–14.

[9] M. Uschold and M. Gruninger, "Ontologies: principles, methods and applications," *The Knowledge Engineering Review*, vol. 11, no. 2, p. 93–136, 1996.

[10] F. Ciroku, "Supporting requirement elicitation and ontology testing in knowledge graph engineering," 2023.

[11] L. Halilaj, N. Petersen, I. Grangel-González, C. Lange, S. Auer, G. Coskun, and S. Lohmann, "Vocol: An integrated environment to support version-controlled vocabulary development," in *European Knowledge Acquisition Workshop*. Springer, 2016, pp. 303–319.

[12] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Perez, A. Fernández-Izquierdo, and O. Corcho, "Automating ontology engineering support activities with ontoology," *Journal of Web Semantics*, vol. 57, p. 100472, 2019.

[13] N. Matentzoglu, C. Mungall, and D. Goutte-Gattat, "Ontology development kit," Jul. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.6257507

[14] R. C. Jackson, J. P. Balhoff, E. Douglass, N. L. Harris, C. J. Mungall, and J. A. Overton, "Robot: a tool for automating ontology workflows," *BMC bioinformatics*, vol. 20, no. 1, pp. 1–10, 2019.

[15] D. Allemang, P. Garbacz, P. Gradzki, E. Kendall, and R. Trypuz, "An infrastructure for collaborative ontology development, lessons learned from developing the financial industry business ontology (FIBO)," in *Formal Ontology in Information Systems*. IOS Press, 2022.

[16] S. Bader, J. Pullmann, C. Mader, S. Tramp, C. Quix, A. W. Müller, H. Akyürek, M. Böckmann, B. T. Imbusch, J. Lipp *et al.*, "The international data spaces information model–an ontology for sovereign exchange of digital content," in *ISWC 2020, International Semantic Web Conference, Athens, November 2–6*. Springer, 2020, pp. 176–192.

[17] R. García-Castro, M. Lefrançois, M. Poveda-Villalón, and L. Daniele, *The ETSI SAREF Ontology for Smart Applications: A Long Path of Development and Evolution*. Wiley-IEEE Press, 2023, pp. 183–215.

[18] M. Lefrançois and D. Gnabasik, "The saref pipeline and portal—an ontology verification framework," in *International Semantic Web Conference*. Springer, 2023, pp. 134–151.

[19] G. C. Publio, J. E. L. Gayo, G. F. Colunga, and P. Menendéz, "Ontoloci: Continuous data validation with shex," in *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022)*, 2022.

[20] M. Lanthaler, D. Wood, and R. Cyganiak, "RDF 1.1 concepts and abstract syntax," W3C, W3C Recommendation, Feb. 2014, https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[21] B. Parsia, P. Patel-Schneider, S. Rudolph, P. Hitzler, and M. Krötzsch, "OWL 2 web ontology language primer (second edition)," W3C, W3C Recommendation, Dec. 2012, https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[22] A. Seaborne and S. Harris, "SPARQL 1.1 query language," W3C, W3C Recommendation, Mar. 2013, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[23] D. Kontokostas and H. Knublauch, "Shapes constraint language (SHACL)," W3C, W3C Recommendation, Jul. 2017, https://www.w3.org/TR/2017/REC-shacl-20170720/.

[24] S. Sahoo, T. Lebo, and D. McGuinness, "PROV-o: The PROV ontology," W3C, W3C Recommendation, Apr. 2013, https://www.w3.org/TR/2013/REC-prov-o-20130430/.

[25] S. Abou-Zahra, "Evaluation and report language (EARL) 1.0 schema," W3C, W3C Note, Feb. 2017, https://www.w3.org/TR/2017/NOTE-EARL10-Schema-20170202/.

[26] R. Cérès, O. Corby, E. Demairy, and F. Gandon, "Corese," Jul. 2023. [Online]. Available: https://hal.science/hal-04170333

[27] T. Berners-Lee, R. T. Fielding, and L. M. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, Jan. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc3986

[37] https://github.com/Wimmics/olivaw