

The ACIMOV Methodology: Agile and Continuous Integration for Modular Ontologies and Vocabularies

Fatma-Zohra Hannou^{1,*}, Victor Charpenay¹, Maxime Lefrançois¹,
Catherine Roussey², Antoine Zimmermann¹ and Fabien Gandon³

¹Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, Saint-Étienne, France

²MISTEA, INRAE, Place Pierre Viala, Montpellier, France

³Inria, Univ. Cote d'Azur, I3S, CNRS, France

Abstract

This work describes the Agile and Continuous Integration for Modular Ontologies and Vocabularies (ACIMOV) ontology engineering methodology for developing ontologies and vocabularies. ACIMOV extends the SAMOD agile methodology to (1) ensure alignment to selected reference ontologies; (2) plan module development based on dependencies; (3) define ontology modules that can be specialized for specific domains; (4) empower active collaboration among ontology engineers and domain experts; (5) enable application developers to select views of the ontology for their specific domain and use case. ACIMOV adopts the standard git-based approach for coding, leveraging agility and DevOps principles. It has been designed to be operationalized using collaborative software development platforms such as Github or Gitlab, and toolled with continuous integration and continuous deployment workflows (CI/CD workflows) that run syntactic and semantic checks on the repository, specialize modules, generate and publish the ontology documentation.

Keywords

Modular ontologies, Ontology engineering methodology, Agile method, Git, Continuous Integration and Continuous Deployment

1. Introduction

As formal and structured representations of knowledge, ontologies are key software assets for achieving semantic interoperability among complex systems such as Internet of Things (IoT) applications, industrial control systems, and robotics.

Monolithic ontologies are difficult to maintain, hard to understand and adopt and therefore serve little to other domains and applications besides those for which they were initially created. Modularity in ontology design refers to breaking down knowledge into smaller pieces called

2nd Workshop on Modular Knowledge, 9th Joint Ontology Workshops (JOWO 2023), co-located with FOIS 2023, 19-20 July, 2023, Sherbrooke, Québec, Canada


*Corresponding author.

✉ fatma-zohra.hannou@emse.fr (F. Hannou); victor.charpenay@emse.fr (V. Charpenay);
maxime.lefrancois@emse.fr (M. Lefrançois); catherine.roussey@inrae.fr (C. Roussey);
antoine.zimmermann@emse.fr (A. Zimmermann); fabien.gandon@inria.fr (F. Gandon)

🆔 0000-0003-4747-1232 (F. Hannou); 0000-0002-9210-1583 (V. Charpenay); 0000-0001-9814-8991 (M. Lefrançois);
0000-0002-3076-5499 (C. Roussey); 0000-0003-1502-6986 (A. Zimmermann); 0000-0003-0543-1232 (F. Gandon)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

modules or sub-ontologies. Hence, a modular ontology is constructed as a set of modules by connecting concepts relevant to the domain or application at hand.

Building an ontology from small modules that can be specialized for different domains brings understandability and homogeneity to the overall ontology. Modular ontologies based on such module specializations enable developers of specific applications to create custom views by selecting the set of required specializations. The underlying knowledge factorization also streamlines the development process and eases maintainability and resilience, considering that problems arising within a module are handled in a targeted fashion.

Ontology development methodologies play a determining role in the development of high-quality ontologies that satisfy application requirements. As ontologies are software resources, advances in software engineering best practices, such as agility, version control management, and DevOps, can be transposed to ontology engineering. Ontology engineering literature has dedicated extensive work to methodologies definition, many of them adapting some agile methodology. In this paper, we propose to extend existing agile methodologies with git-based development tools in the case of modular ontologies collaborative design.

Accordingly, the paper proposes an approach to collaboratively develop modular ontologies for multi-domain applications, by adapting the standard git-based approach for coding.

The rest of this paper is structured as follows. Section 2 introduces the research project in which this work has been led, and the resulting requirements for the ontology and its development methodology. Section 3 overviews related work on designing modular IoT domain ontologies. Section 4 describes the ACIMOV methodology, designed as an extension of the Simplified Agile Methodology for Ontology Development (SAMOD) [1]. Section 5 describes how ACIMOV can be operationalized with collaborative software development platforms, and tooled with scripts.

2. Context and Requirements

The Constrained Semantic Web of Things (CoSWoT) project [2] investigates the exchange of semantic data and the distribution of semantic reasoning in resource-constrained execution environments, which are common in the internet of things (IoT) applications. It aims at providing domain-independent solutions and focuses on two vertical domains: smart buildings and precision farming. Use cases for the building domain include: (SB1) personalized dashboards containing aggregated sensor measurements and enabling automatic alerts, (SB2) heating and window-opening control for optimizing the occupant's health, comfort, and energy consumption. Use cases for the precision farming domain include: (PF1) estimate the crop's growth stages or weather events based on outdoor air temperature measurements, (PF2) planning watering based on soil moisture level, crop's growth stage, and rainfall measurements, (PF3) speed adaptation of autonomous machines to fields' moisture measurements.

The knowledge model for the CoSWoT project must hold knowledge common to all IoT platform components, and knowledge specific to some domain or application. Subsets of the CoSWoT ontology, or *views*, will ultimately be embedded in resource-constrained devices for representing, reasoning, and exchanging sensor data. We summarize the requirements identified for the CoSWoT ontology as follows:

- O1** The ontology must align to reference IoT ontologies;
- O2** The ontology must be modular, including modules that cover knowledge common to all IoT platform components;
- O3** The ontology must reuse some identified ontologies for the application domains at stake;
- O4** The ontology must have a homogeneous and predictable structure, such that similar concepts for different domains are described the same way;
- O5** Different alternative representations must be possible to account for the need to manipulate small knowledge graphs in constrained devices;
- O6** One must be able to select a subset of the ontology (a view) that covers the needs of a specific application.

In addition to ontology engineers, various stakeholder profiles are involved in the development of the CoSWoT ontology: domain experts (DEs) and end-product owners (POs) actively and continuously collaborate to elicit requirements, develop the ontology, develop applications that use the ontology, and maintain the ontology. In this context, the ontology development methodology should fulfill some requirements, to ensure good communication and maximize the effectiveness and efficiency of the collaborative effort.

- M1** Agile principles must be adopted to improve collaboration between ontology engineers, domain experts, and end-product owners, with short cycles, and working increments;
- M2** Regular meetings with all parties must be held to help prioritizing the requirements stemming from use cases, and choosing the target for the next iteration;
- M3** Regular meetings among ontology engineers must be held, to help prioritizing the modules to work on, and ensuring work on different modules can be led in parallel;
- M4** Collaborative software development platforms with code versioning and issue tracking shall be adopted;
- M5** DevOps principles must be adopted to enable continuous integration and deployment of the ontology artifacts (e.g., ontology modules, documentation, examples)

3. Related Work for Designing Modular IoT Domain Ontologies

3.1. Modular IoT ontologies

Three IoT ontologies can be considered as references from standardization bodies: W3C Thing Description (TD) [3], OGC&W3C Semantic Sensor Network (SSN) [4], ETSI Smart Applications Reference ontology (SAREF) [5]. These three ontologies adopt some form of modular design. TD comes hand-to-hand with the Hypermedia Controls Ontology (HCTL) [6] and the JSON Schema in RDF vocabulary [7], and relies on separate vocabularies for protocol bindings, such as for HTTP, COAP, and MQTT [8]. SSN consists of a lightweight core called SOSA (Sensor, Observation, Sample, and Actuator), a more expressive extension module SSN, a separate module SSN-Systems for system capabilities, and a set of alignment modules to other ontologies. SAREF consists of a core ontology SAREF Core, and different extensions for verticals including SAREF4BLDG and SAREF4AGRI for building and agriculture domains.

3.2. Agile ontology engineering methods

Many ontology engineering methodologies have been proposed over time, including METHONTOLOGY [9], On-To-Knowledge [10], DILIGENT [11], the “Ontology Development 101” [12], NeOn [13]. Some directly transpose software engineering methodologies, for example UPON Lite [14] is based on Rational Unified Process. The LOT methodology [15] adopts a V-model approach with conditional feedback at upstream development stages. Other early methods proposed to rely and align with existing ontologies to bootstrap new ontologies as in SENSUS [16].

More recently, methodologies are inspired by the principles of Agile software engineering, which promote collaboration between developers and stakeholders by producing regular updates of the product¹. Among these methods, AMOD [17] and CD-OAM [18] are based on SCRUM. AMOD is the first method that describes the cycle of ontology development in a SCRUM sprint. CD-OAM enriches AMOD by describing the management the ontology commitment user community. XPOD [19] and eXtreme ontology method [20] are based on eXtreme Programming. The Lean Ontology Development (LOD) [21] is inspired by the Lean approach: Build-Measure-Learn. SAMOD [1] is revisiting the motivating scenarios and competency questions of Uschold and Gruninger [22], additionally considering ontology modules and test-driven development.

3.3. Git and CI/CD for ontology engineering

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 task pipelines that will be executed automatically when, for example, a commit is pushed to the server. Before the democratization of these frameworks, a few preliminary approaches such as VoCol [23] or OnToology [24] were proposed in the ontology engineering community using Github applications². *Ontology Development Kit* (ODK) [25] uses Travis CI to run workflows with the ROBOT tool [26] developed by the Open Biological and Biomedical Ontologies (OBO) community. CI/CD pipelines are reported for the publication of different ontologies, such as the Financial Industry Business Ontology (FIBO) in [27], the International Data Spaces Information Model (IDSA) in [28], and the CASE Cyber Ontology³. Specific Github actions are available on the Github marketplace for running RDFLint⁴, validating RDF syntaxes^{5,6}, or validating RDF files against SHACL shapes⁷ or ShEx [29].

3.4. Positioning the Contributions of the CoSWoT project

Upfront, we have chosen to align the CoSWoT ontology to the three reference IoT ontologies TD, SSN, SAREF. These ontologies cover partially overlapping requirements and sometimes

¹The Manifesto for Agile Software Development lists 12 principles. <http://agilemanifesto.org/iso/en/principles.html>

²<https://docs.github.com/en/developers/apps>

³<https://github.com/marketplace/actions/case-ontology-validator>

⁴<https://github.com/marketplace/actions/setup-rdfint>

⁵<https://github.com/marketplace/actions/rdf-syntax-check>

⁶<https://github.com/marketplace/actions/validate-rdf-with-jena>

⁷<https://github.com/marketplace/actions/validate-shacl>

adopt different modeling choices, which makes fulfilling Requirement **O1** a challenging task. In addition, even in their communities of developers there are sometimes disagreements on how concepts should be used.⁸

The SAMOD methodology covers requirement **M1**. It is an agile method that promotes the incremental development of the ontology based on the aggregation of small and simple ontology chunks named *modelets*, which can be seen as implementations of design pattern. SAMOD encourages the reuse of existing ontology design patterns when relevant, instead of entire ontologies. The resulting ontology (current final model in SAMOD) is built incrementally and sequentially, one modelet by one modelet. SAMOD does not describe how modelets may be reworked (Requirement **M2**), nor how their development can be prioritized and led in parallel by different ontology engineer (Requirement **M3**). Section 4 describes the ACIMOV methodology as an extension of SAMOD that covers requirements **O1**, **O2**, **O5**, and **M1** to **M3**.

To the best of our knowledge, the use of SAMOD has only partially been operationalized with the Github collaborative software development platform and CI/CD workflows, for instance in the Poliphonia project⁹, and this aspect has not been published yet (Requirements **M4** and **M5**). Section 5 describes how we defined similar pipelines as those used on the FIBO, IDSA and CASE ontologies, to automate testing and publishing. In addition, it describes how we leverage DevOps tools to encourage ontology engineers to follow the methodology.

Finally, SAMOD may help covering Requirements **O2** and **O5** considering each modelet is a module, but requirements **O4** and **O6** are not covered. Section 5 describes how the ontology may consist of modelets that can be specialized for different domains.

4. Description of the ACIMOV Methodology

Figure 1 provides an overview of the seven steps in the ACIMOV methodology. The methodology consists of an outer, longer cycle that involves ontology engineers, domain experts, and end-product owners, and two inner, shorter cycles dedicated to development activities carried out by ontology engineers. The steps are:

- Step 1** Collect requirements and identify reference ontologies.
- Step 2** Review meeting (an event)
- Step 3** Select relevant modules from reference ontologies
- Step 4** Manage modelet backlog
- Step 5** Modelet development meeting (an event)
- Step 6** Develop and test modelets
- Step 7** Integrate modelet and release ontology artifacts

The rest of this section describes these main steps, events, and artifacts, formalized in ACIMOV.

⁸See for example the debate about whether instances of `ssn:Property` should be specific to some feature of interest, or generic. [4, Footnote 8] and https://www.w3.org/2015/spatial/wiki/What_is_an_instance_of_ssn:Property.

⁹<https://github.com/poliphonia-project/stories>

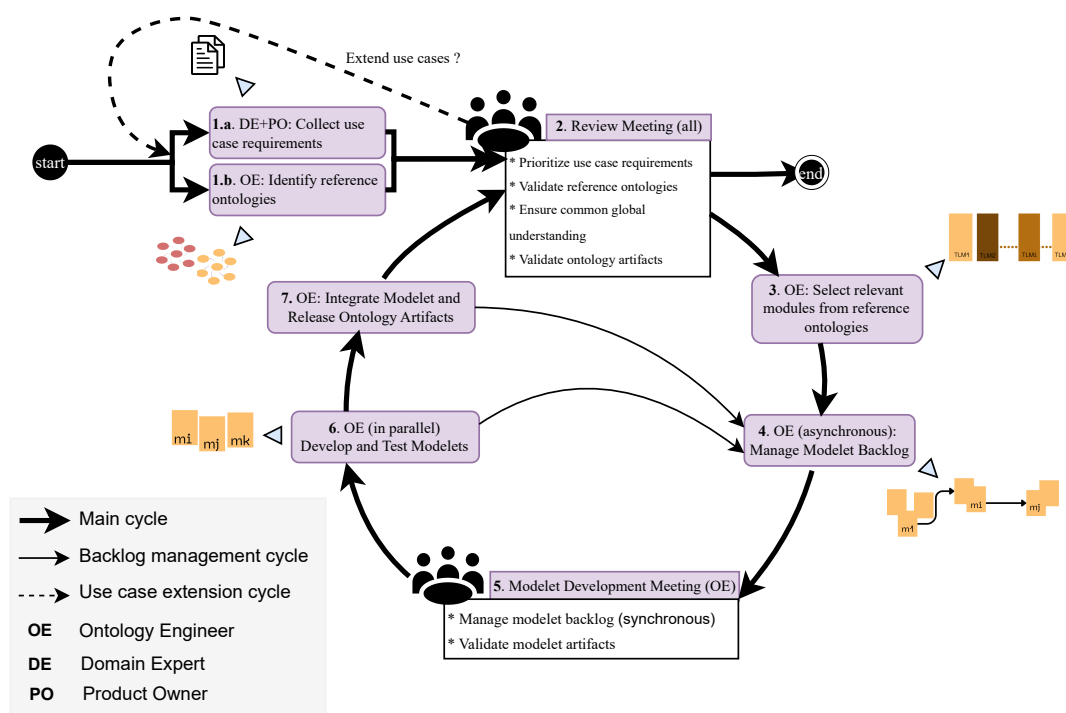


Figure 1: Overview of the ACIMOV methodology

4.1. Step 1: Collect Requirements and Identify Reference Ontologies

The outer cycle in ACIMOV starts with the collection of use cases requirements and the identification of reference ontologies. Requirement collection aims to determine the extent of the knowledge to capture. Domain experts autonomously specify use cases, and end-product owners define the application scope and how software components will make use of the ontology. In parallel, ontology engineers identify reference IoT ontologies (Req. **O1**) and reference ontologies for the domains at stake (Req. **O3**). These will facilitate the ontology development process and improve interoperability and community adoption.

4.2. Step 2: Review Meeting

After **Step 1**, a *review meeting* (Req. **M2**) is organized to specify and prioritize use case requirements, validate the selection of reference ontologies, and ensure a common global understanding of the domain and the end-products. Accordingly, ontology developers formulate a set of competency questions (CQs) that the ontology should be able to answer. An example of CQs for a module in the CoSWoT project can be found in the project repository¹⁰.

In subsequent development cycles, review meetings are also organized when the current version of the *ontology artifacts* is presented, reviewed, and validated with respect to its adequacy to the use cases and the end-product applications.

¹⁰<https://gitlab.com/coswot/coswot-samod/-/blob/master/domains/building-automation/evaluation/README.md>

Ontology artifacts include the set of modules, the ontology and its documentation, validation reports, and examples.

4.3. Select Relevant Modules from Reference Ontologies

Based on the output of a review meeting, ontology engineers examine how subsets of the selected reference ontologies may wholly or partially match the requirements, and address potential modeling discrepancies. In the case of different modeling choices, reconciliation involves identifying the most suitable to the application at hand or merging complementary modules to enlarge the set of covered concepts. For example, the IoT ontologies SSN and SAREF both describe sensor observations/measurements, but with different modeling choices. Documented ontology design patterns may help to find a common ground or generalization in case of modeling discrepancies. As an alternative, ontology engineers may conduct a deep analysis of the reference ontologies, using some upper ontology such as the Dolce-Ultralite ontology (DUL) [30].

The output of this task is reference ontology modules that will be reused in the ontology development. Some of them may have the potential of being specialized in different domains. For example, the *sensor* class could be specialized as *moisture sensor*, *temperature and relative humidity sensor*, or *temperature sensor*. Reference ontology modules keep track of the provenance to the reference ontology(ies) they are selected from, thus facilitating semantic interoperability with applications that use these reference ontologies. Selected reference ontology modules for CoSWoT describe *sensors*, *affordances*, *quantity kinds*, ...The full list can be found on the project repository.

4.4. Step 4: Manage Modelet Backlog

To address modularity needs (Req. O2), the ontology is designed as a set of stand-alone modules called *modelets*, each describing a particular aspect of the ontology and covering a small set of requirements. Modelets have a loose N-N correspondence with reference ontology modules, allowing some flexibility and opening up extension tracks.

For requirement O4, modelets are organized and managed in a *backlog*, with some relations such as *isSpecializationOf*, *dependsOn*, *isAlternativeFor*, *hasHigherPriorityThan*.

The coverage of the current list of requirements can be assessed based on the list of modelets in the backlog. The network of modelets can be analyzed to select the next top-priority ones, or those on which depend top-priority modelets. As an output of this task, some modelets are assigned to ontology engineers for the next session of collaborative and parallel development.

4.5. Step 5: Modelet Development Meeting

Part of the backlog management and modelet assignment is done asynchronously through an issue tracker. In addition, dedicated *modelet development meetings* among ontology engineers (OEs) are organized (Req. M3). These meetings are the starting point of the inner cycles in ACIMOV. In subsequent iterations of the inner cycles, modelet development meetings are when the current version of the *modelet artifacts* are presented and reviewed, backlog change requests are raised and discussed, and modeling choices for dependent modelets are synchronized.

4.6. Step 6: Develop and Test Modelets

Each ontology engineer is in charge of developing and testing a modelet's artifacts. A typical modelet definition includes the following elements:

Description: motivation statement extracted from use case scenarios of one or more domains.

Competency questions: a list of questions that translate the requirements, and examples of responses in natural language;

Glossary: optional correspondences to reference ontology modules.

Diagrams: CHOWLK [30] diagram that synthesizes the elements of the modelet. Diagrams play a central role as they facilitate team discussions.

Ontological description: specification of classes, properties and individuals introduced by this modelet.

Example: some RDF graph instantiating the modelet for the use case scenarios.

Example SPARQL queries: formalization of competency questions as SPARQL queries, that may be executed over the examples.

Recommendations: how to best use the modelet in other modelets or applications. This includes entity naming conventions to improve clarity and consistency.

Scripts: any automated script that can be run to specialize this modelet for different applications, or check that specializations satisfy the recommendations.

A modelet can be developed from scratch, following a formalization process where the terms in the requirements are mapped to classes and properties, and use case constraints are mapped to axioms. Alternatively a modelet can be adapted from an existing one, leveraging:

Generalization. For example, *SSN Observation*, *Sampling*, and *Actuation*, may be generalized with some new class *ProcedureExecution* that generically describes the act of executing some procedure, like observation or actuation.

Specialization. For example, *ProcedureExecution* can be specialized as *ProcedureExecutionOnProperty*, with additional object properties that link to the *Property* over which the activity is carried out, and the result. The resulting modelet better covers *Observation*, *Actuation*, and additionally covers forecasts, planned actuations, etc.

Domain-Specific specialization. Top-domain classes or properties can be specialized for vertical domains (Req. O6). For example *FeatureOfInterest* may be specialized as *Office* and *MeetingRoom*, but also *SoilOfPlot* or *CropCultivatedInThePlot*. *Property* may be specialized as *IndoorAirTemperature* and *CarbonDioxideConcentration*, but also *MoistureLevel* and *CropDevelopmentStage*.

Alternative. Modelet may introduce alternative modeling choices (Req. O5). For example, as an alternative to modeling results of procedure executions using the QUDT ontology, one may introduce datatype properties such as *hasSimpleResultInDegreesCelsius* to concisely link an observation to a decimal that encodes its result with the degrees Celsius unit.

Modelet *testing* requires checking the quality and coherence of the ontological description, examples of SPARQL queries, and scripts. Modelet *validation* requires successful testing results (eg. competency questions), and a decision from the team of ontology engineers during some

modelet development meeting. This can be obtained after looping back to **Step 4**, then a next session of collaborative and parallel development may be planned.

When a modelet is validated, one may proceed with the modelet integration and release preparation in **Step 7**.

4.7. Step 7: Integrate Modelet and Release Ontology Artifacts

Modelets that are validated can be integrated into the global ontology. This requires global checks such as to ensure the absence of name clashes, and a careful review of dependencies that link the modelet to other modelets, so as to mention interrelation when appropriate. As part of this last step, the global ontology documentation can be improved, and the ontology artifacts can be produced and released, with respect to DevOps best practices (Req. **M5**).

The ontology engineers may proceed with the management of the modelet backlog (**Step 4**), and a modelet development meeting to plan the next session of collaborative and parallel development. Alternatively, the ontology artifacts may be reviewed and possibly validated by all the stakeholders during a review meeting, opening up three different possibilities:

Loop to Step 1 if more work is needed by the domain experts and the end-product owners to extend or refine the set of requirements;

Proceed with Step 3 if a new set of use case requirements is selected to carry on the development of the ontology;

End if the ontology is considered done.

5. Operationalization of the ACIMOV Methodology

The ACIMOV methodology aims to be aligned with modern collaborative software development platforms (Req. **M4**), such as Github and Gitlab. These platforms provide Git-based code versioning, integrate project management tools such as issue trackers and Kanban boards, allow for release hosting, and provide a range of continuous integration and continuous deployment (CI/CD) features.

Github and Gitlab come with a canonical workflow that drives the way various software management features are integrated into a single platform. Both workflows¹¹ are themselves based on the default Git branching workflow¹². Git's versioning model is a directed graph of changes (additions/deletions of lines of text), also known as *commits*. A commit can have multiple children, each materializing a *branch*, and multiple parents, if some branch is merged into another one. The default Git workflow is defined upon three kinds of branches:

- *main* branches, which include stable, successive versions of a code base;
- *feature* (a.k.a. *topic*) branches, which branch a code base to implement a new feature or to improve or fix an existing feature;
- *development* branches, which integrate the content of (potentially conflicting) feature branches before merging the in-development code base into a main branch.

¹¹<https://guides.github.com/introduction/flow/>, https://docs.gitlab.com/ee/topics/gitlab_flow.html

¹²<https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>

ACIMOV naturally aligns with the default Git branching workflow. Modelets are developed in feature branches in **Step 6**, and merged into the main branch in **Step 7**. Several feature branches may be created and independent modelets can be developed in parallel. The workflow consists of the following steps:

1. create a new feature branch from the reference main or a development branch
2. commit one or more changes to the feature branch
3. perform unit tests on the feature being implemented or fixed
4. merge feature branch into the development branch
5. perform integration tests
6. merge development branch into main branch

During the methodology **Step 6**, CI/CD features can be leveraged to automate the execution of modelet testing during their development in feature branches. Custom syntactic and semantic tests are executed before committing to a feature branch or after merging to the main branch. Their integrated issue tracker is also commonly used to document desired or in-development features, in that an issue can be tied to a specific feature branch: once the issue is created, a branch is also created and once the branch is merged, the issue is closed. The application of the default Git workflow to ontology engineering is straightforward, once a correspondence to topics or features has been found. Our methodology stipulates that changes in the ontology are performed per modelet. Creating a new modelet should thus be performed in a dedicated feature branch. The “modelet branch” should in turn only be merged if the competency questions of the modelet are fully addressed and tested. The content produced in a modelet (formal content in RDF, OWL, SHACL, SPARQL or other languages, query answers in CSV files, or any other textual and graphical documentation) should be isolated from the rest of the code base. Upon merging the created or modified modelet, a continuous integration pipeline compiles it into a publishable version, along with existing modelets.

A summary of developed and suggested integration scripts is given in Table 1.

Workflow step	Script
Step 2	check syntax of README in modelet check syntax of Turtle files in modelet check syntax of SPARQL files in modelet check syntax of CSV files in modelet
Step 4	check that modelet has a title and a description check that (at least) one term is correctly defined in modelet check that (at least) one CQ is correctly defined in modelet evaluate tests in modelet generate documentation of snapshot
Step 6	check that all terms are introduced in (at least) a modelet generate and release documentation

Table 1: Automated checks and generation tasks

In addition to integration scripts, the operationalization of the ACIMOV methodology includes

a set of scripts to automate the ontology generation based on modelet integrations (Req. **M5**). In CoSWoT, two script types have been defined:

Specialization scripts enable domain experts to derive domain ontologies tailored to specific application requirements from the core ontology (Req. **M1** and **O6**). This automation is simplified by the use of *YAML* files to support domain experts in specifying classes and properties configuration without extensive coding. An example of a specialization script and *YAML* file are available on the project repository.

Consolidation scripts minimize manual efforts to create views of the ontology constructed from core modelets (Req. **O6**). Scripts integrate the management of dependencies to guarantee the validity and consistency of generated views and ensure the integration of the corresponding annotations and documentation.

6. Conclusion

This paper introduces the ACIMOV ontology development methodology extending SAMOD by leveraging git-based collaborative development solutions. The methodology is designed for modular ontologies development with a focus on reusing reference ontology modules. It consists of seven steps covering two development cycles carried out by ontology engineers for backlog management and a long collaborative cycle where domain experts and end-product owners feed the requirements collection and participate in the validation process. The methodology has been operationalized using Gitlab as a collaborative software development platform, integrating scripts for automatic domain-driven ontologies generation.

We have several perspectives for this work. First we are reproducing the methodology in the context of two other projects (HyperAgents¹³, ACCORD¹⁴) which are using Github as their Ontology project management platform. We plan to use these second experimentations as evaluation. Then we intend to provide guidelines, templates, and scripts for the operationalization of ACIMOV on Gitlab and Github leveraging and specializing all the functionalities they provide to the case of ontology project management and development.

Acknowledgements

This work has been partly supported by grants from the French research agency (ANR) on projects CoSWot (ANR-19-CE23-0012) and HyperAgents (ANR-19-CE23-0030), and EU project ACCORD (Horizon Europe R&I programme grant agreement no. 101056973).

References

- [1] S. Peroni, Samod: an agile methodology for the development of ontologies, in: Proceedings of the 13th OWL: Experiences and Directions Workshop and 5th OWL reasoner evaluation workshop (OWLED-ORE 2016), 2016, pp. 1–14.

¹³<https://www.hyperagents.org/>

¹⁴<https://accordproject.eu/>

- [2] F. Antoniazzi, G. Ateazing, F. Badeig, M. Bennara, S. Bernard, P. Champin, J. Chanet, C. Gravier, Y. Gripay, F. Laforest, M. Lefrançois, L. Médini, L. Moiroux, C. Roussey, S. Servigne, K. Singh, J. Subercaze, A. Zimmermann, Interopérabilité et raisonnement dans le web sémantique des objets: le projet coswot, in: S. Ferré (Ed.), IC 2020 : 31es Journées francophones d'Ingénierie des Connaissances (Proceedings of the 31st French Knowledge Engineering Conference), Angers, France, June 29 - July 3, 2020, 2020, pp. 155–158. URL: https://hal.archives-ouvertes.fr/IC_2020/hal-02888216.
- [3] V. Charpenay, M. Lefrançois, M. Poveda Villalón, S. Käbisch, Thing Description (TD) Ontology, Editor draft, 10 May 2023, W3C Working Group draft, W3C, 2023.
- [4] A. Haller, K. Janowicz, S. J. Cox, M. Lefrançois, K. Taylor, D. Le Phuoc, J. Lieberman, R. García-Castro, R. Atkinson, C. Stadler, The sosa/ssn ontology: a joint w3c and ogc standard specifying the semantics of sensors, observations, actuation, and sampling, *Semantic Web-Interoperability, Usability, Applicability an IOS Press Journal* 56 (2019).
- [5] R. García-Castro, M. Lefrançois, M. Poveda-Villalón, L. Daniele, The ETSI SAREF ontology for smart applications: a long path of development and evolution, in: A. Moreno-Munoz, N. Giacomini (Eds.), *Energy Smart Appliances: Applications, Methodologies, and Challenges*, Wiley, 2023.
- [6] V. Charpenay, M. Kovatsch, Hypermedia Controls Ontology, Editor draft, 10 May 2023, W3C Working Group draft, W3C, 2023.
- [7] V. Charpenay, M. Lefrançois, M. Poveda Villalón, JSON Schema in RDF, Editor draft, 10 May 2023, W3C Working Group draft, W3C, 2023.
- [8] M. Koster, E. Korkan, Web of Things (WoT) Binding Templates, W3C Working Group Note, 30 January 2020, W3C Working Group Note, W3C, 2020.
- [9] M. Fernández-López, A. Gómez-Pérez, N. Juristo, *Methontology: from ontological art towards ontological engineering* (1997).
- [10] S. Staab, R. Studer, H.-P. Schnurr, Y. Sure, Knowledge processes and ontologies, *IEEE Intelligent systems* 16 (2001) 26–34.
- [11] H. S. Pinto, S. Staab, C. Tempich, Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies, in: *ECAI*, volume 16, Citeseer, 2004, p. 393.
- [12] N. F. Noy, D. L. McGuinness, et al., *Ontology development 101: A guide to creating your first ontology*, 2001.
- [13] A. Gómez-Pérez, M. C. Suárez-Figueroa, Neon methodology: scenarios for building networks of ontologies, in: *16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns (EKAW 2008)*. Conference Poster, 2008.
- [14] A. De Nicola, M. Missikoff, A lightweight methodology for rapid ontology engineering, *Communications of the ACM* 59 (2016) 79–86.
- [15] M. Poveda-Villalón, A. Fernández-Izquierdo, M. Fernández-López, R. García-Castro, Lot: An industrial oriented ontology engineering framework, *Engineering Applications of Artificial Intelligence* 111 (2022) 104755.
- [16] W. Swartout, R. Patil, K. Knight, T. Russ, Toward distributed use of large-scale ontologies, in: *Proceedings of the 10th Banff Knowledge Acquisition Workshop*; Banff, Alberta, Canada; November 9-14, 1996, 1996, p. 11.
- [17] A. S. Abdelghany, N. R. Darwish, H. A. Hefni, An agile methodology for ontology develop-

- ment, *International Journal of Intelligent Engineering and Systems* 12 (2019) 170–181.
- [18] A. Takhom, S. Usanavasin, T. Supnithi, P. Boonkwan, A collaborative framework supporting ontology development based on agile and scrum model, *IEICE TRANSACTIONS on Information and Systems* 103 (2020) 2568–2577.
- [19] A. A. Sharifloo, M. Shamsfard, Using agility in ontology construction., in: *Formal Ontologies Meet Industry*, volume 174 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2008, pp. 109–119.
- [20] M. Hristozova, 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.
- [21] J. Cummings, D. Stacey, Lean ontology development: An ontology development paradigm based on continuous innovation., in: *Knowledge Engineering and Ontology Development*, 2018, pp. 365–372.
- [22] M. Uschold, M. Gruninger, Ontologies: principles, methods and applications, *The Knowledge Engineering Review* 11 (1996) 93–136. doi:10.1017/S0269888900007797.
- [23] L. Halilaj, N. Petersen, I. Grangel-González, C. Lange, S. Auer, G. Coskun, S. Lohmann, Vocol: An integrated environment to support version-controlled vocabulary development, in: *European Knowledge Acquisition Workshop*, Springer, 2016, pp. 303–319.
- [24] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Perez, A. Fernández-Izquierdo, O. Corcho, Automating ontology engineering support activities with ontoology, *Journal of Web Semantics* 57 (2019) 100472.
- [25] N. Matentzoglu, C. Mungall, D. Goutte-Gattat, Ontology development kit, 2021. doi:10.5281/zenodo.6257507.
- [26] R. C. Jackson, J. P. Balhoff, E. Douglass, N. L. Harris, C. J. Mungall, J. A. Overton, Robot: a tool for automating ontology workflows, *BMC bioinformatics* 20 (2019) 1–10.
- [27] D. Allemang, P. Garbacz, P. Gradzki, E. Kendall, 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.
- [28] 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: *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference*, Athens, Greece, November 2–6, 2020, Proceedings, Part II, Springer, 2020, pp. 176–192.
- [29] G. C. Publio, J. E. L. Gayo, G. F. Colunga, P. Menéndez, Ontolo-ci: 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.
- [30] S. Chávez-Feria, R. García-Castro, M. Poveda-Villalón, Chowlk: from uml-based ontology conceptualizations to owl, in: *The Semantic Web: 19th International Conference, ESWC 2022*, Hersonissos, Crete, Greece, May 29–June 2, 2022, Proceedings, Springer, 2022, pp. 338–352.