Apports des méthodologies et techniques de développement logiciel pour l'ingénierie des ontologies: Retour d'expérience des contributions au développement de l'ontologie ETSI SAREF

Maxime Lefrançois, Raúl García-Castro, María Poveda-Villalón, Omar Qawasmeh

Apports des méthodologies et techniques de développement logiciel pour l'ingénierie des ontologies: Retour d'expérience des contributions au développement de l'ontologie ETSI SAREF

Maxime Lefrançois[1], Raúl García-Castro[2], María Poveda-Villalón[2], Omar Qawasmeh[3]

[1] Mines Saint-Étienne, Univ Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, F - 42023 Saint-Étienne France
[2] Open Engineering Group, Universidad Politécnica de Madrid, Spain
[3] Hybrid Intelligence, Capgemini Engineering, 69007 Lyon, France

maxime.lefrancois@emse.fr, rgarcia@fi.upm.es, mpoveda@fi.upm.es, omar.alqawasmeh@capgemini.com

Résumé

L'ingénierie logicielle a toujours eu une grande influence dans l'ingénierie des ontologies. Cet article a pour objectif d'identifier ces influences pour certains des grands thèmes de l'ingénierie logicielle moderne : 1. Ingénierie des besoins ; 2. Modèles de cycle de vie du développement logiciel ; 3. Modularisation ; 4. Patrons ; 5. Environnements de développement ; 6. Nommage des versions ; 7. Contrôle des versions et workflow d'édition ; 8. Automatisation ; 9. Intégration et déploiement continu. Pour chaque thème nous identifions des travaux du domaine de l'ingénierie des ontologies qui s'y rapportent, et apportons un retour d'expérience de notre travail de spécification du cadre de développement et du flux de travail de l'ontologie ETSI Smart Applications REFerence (SAREF), et développement du portail communautaire SAREF.

Mots-clés

Ingénierie logicielle, Agile, DevOps, Ingénierie des Ontologies, SAREF

Abstract

Software engineering has always had a strong influence in ontology engineering. This article aims to identify these influences for some of the major themes of modern software engineering : 1. Requirements engineering ; 2. Software development life cycle models ; 3. Modularization ; 4. Patterns ; 5. Development environments ; 6. Version naming ; 7. Version control and editing workflow ; 8. Automation ; 9. Continuous Integration and Deployment. For each theme we identify work in the field of ontology engineering that relates to it, and provide lessons learned from our work on the specification of the ETSI Smart Applications REFerence ontology (SAREF) development framework and workflow, and development of the Community SAREF Portal for user engagement

Keywords

Software Engineering, Agile, DevOps, Ontology Engineering, SAREF

1 Introduction

L'ontologie Smart Applications REFerence (SAREF) est constituée d'un ensemble modulaire d'ontologies versionnées. SAREF a été promue par la Commission européenne en collaboration avec l'Institut européen des normes de télécommunications (ETSI) dans le but de disposer d'un modèle de données commun pour limiter la fragmentation de l'internet des objets (IoT). L'ontologie SAREF est développée au sein du comité technique SmartM2M de l'ETSI, et est destinée à permettre l'interopérabilité entre les solutions de différents fournisseurs et entre divers secteurs d'activité de l'IoT, contribuant ainsi au développement du marché numérique mondial. Dans cet article nous présentons des résultats du projet ETSI *Specialist Task Force* (STF) 578 récemment terminé, intitulé : "Spécification du cadre de développement et du flux de travail de SAREF, et développement du portail communautaire SAREF pour la participation des utilisateurs". Ce projet avait pour objectif de spécifier le cadre de développement de SAREF et le flux de travail pour accélérer le développement de SAREF et de ses extensions, et développer un logiciel qui sera utilisé pour automatiser la génération du contenu du portail de l'ontologie à partir des sources de SAREF sur la forge ETSI `https://saref.etsi.org/sources/` . La vision finale du projet est de faire en sorte que les industriels utilisateurs de SAREF soient capables d'apporter leur contribution à SAREF et de maintenir SAREF, sans nécessiter de compétences poussées en ingénierie des ontologies ni d'un soutien spécial de l'ETSI, mais juste avec une révision des membres de l'ETSI, et en particulier de SmartM2M.

L'ingénierie logicielle a toujours eu une grande influence dans l'ingénierie des ontologies. Cet article a pour objectif d'identifier ces influences pour certains des grands

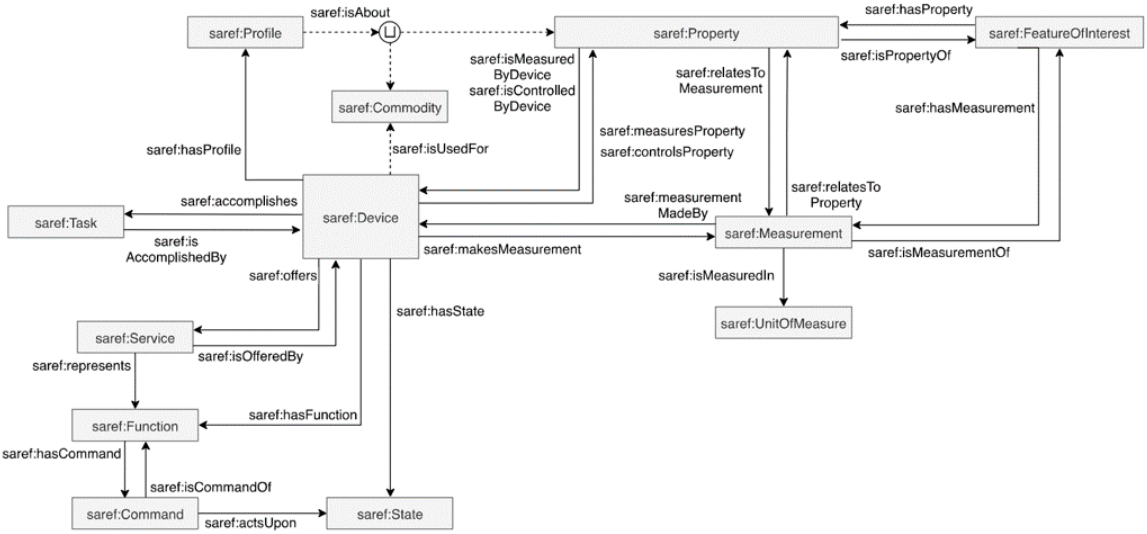

FIGURE 1 – Aperçu des concepts principaux de la version V3.1.1 du module SAREF Core (Source : [23])

thèmes de l'ingénierie logicielle moderne, et d'apporter un retour d'expérience de nos contributions au développement de l'ontologie ETSI SAREF. Nous identifions enfin des difficultés rencontrées ou des directions de travail futures possibles. Nous nous sommes appuyés sur des documents de référence dans le domaine de l'ingénierie logicielle pour sélectionner les thèmes abordés : le *Software Engineering Body of Knowledge (SWEBOK) v3* [36], et le *Systems and Software Engineering Vocabulary (SEVOCAB)* [37]. Ainsi, nous aborderons les thèmes suivants séquentiellement dans cet article : 1. Ingénierie des besoins ; 2. Modèles de cycle de vie du développement logiciel ; 3. Modularisation ; 4. Patrons ; 5. Environnements de développement ; 6. Nommage des versions ; 7. Contrôle des versions et workflow d'édition ; 8. Automatisation ; 9. Intégration et déploiement continu. Il est à noter que de nombreux travaux démontrent les apports des ontologies pour différents sous-domaines de l'ingénierie logicielle, par exemple pour l'élicitation des besoins [40], cependant nous considérons ces travaux hors du cadre de cet article.

2 Ingénierie des besoins

L'ingénierie des besoins en ingénierie logicielle se rapporte à l'élicitation, l'analyse, la spécification, et la validation des besoins logiciels. Dès Grüninger et Fox en 1995 [32], On-To-Knowledge [61] en 2001, eXtreme ontology method [35] en 2002, les méthodologies d'ingénierie des ontologies comportent habituellement une phase de spécification des besoins sous forme de questions de compétences [64], sous une forme textuelle, de logique du premier ordre, ou sous forme d'une requête SPARQL. L'étude des techniques d'élicitation des connaissances pour dériver des questions de compétences a été étudiée par exemple par Rao et. al. [58]. Le concept de *Software Specification Document* a été transposé aux ontologies : le *Ontology Requirement Specification Document* [62]. La thèse récente de Alba Fernández Izquierdo [38] porte sur la spécification et l'évaluation automatique des besoins pour l'ingénierie des ontologies.

Dans le contexte du projet STF 578 de l'ETSI, nous recommandons la présence d'un document spécifiant les besoins pour tout projet d'ontologie SAREF [22, Clause 9], sous forme d'un document CSV avec trois colonnes : un identifiant, une catégorie, et un besoin exprimé sous forme d'une affirmation ou d'une question de compétences. Ces besoins sont ensuite évalués avec l'outil Themis [25].

3 Modèles de cycle de vie du développement logiciel

De nombreuses méthodologies d'ingénierie des ontologies ont été proposées au fil du temps, dont METHONTOLOGY [26], On-To-Knowledge [61], DILIGENT [54], le "Ontology Development 101" [50], NeOn [29]. Certaines transposent directement des méthodologies d'ingénierie logicielle, par exemple UPON Lite [10] est basé sur Rational Unified Process. Les récentes méthodologies s'inspirent toutes des principes d'ingénierie logicielle Agile. AMOD [1] et CD-OAM [63] sont basés sur SCRUM, XPOD [59] et eXtreme ontology method [35] sont basés sur eXtreme Programming, Lean Ontology Development (LOD) [9] s'inspire de l'approche Lean, SAMOD [53] se base sur les concepts de stories, itérations, et développement dirigé par les tests. Le développement des ontologies SAREF suit la méthodologie LOT [55], qui adopte une approche de type V-model avec des retours conditionnels à des étapes de développement amont.

Cette liste n'est pas exhaustive, mais démontre que le domaine de l'ingénierie logicielle inspire et influence celui de l'ingénierie des ontologies. Il est donc pertinent de surveiller les évolutions du premier domaine pour continuer à améliorer le second.

4 Modularisation

La modularité est définie comme *le degré selon lequel un système ou un programme informatique est composé d'éléments distincts, de sorte que la modification d'un élément a un impact minimal sur les autres éléments* [37]. Une transposition aux ontologies et un premier algorithme de modularisation ont été proposés dans [31]. Le sujet intéresse la communauté, voir par exemple les séries de workshops WoMO (Workshop on Modular Ontologies, de 2006 à 2013) et WOMoCoE (Workshop on Ontology Modularity, Contextuality, and Evolution, de 2016 à 2020). Beaucoup d'ontologies sont aujourd'hui publiées sous forme de réseau d'ontologies, composé de modules faiblement dépendants qui utilisent le mécanisme d'import de OWL [48, Sec. 3.4]. Une ontologie noyau n'importe aucune autre ontologie du réseau, des ontologies périphériques importent l'ontologie noyau et potentiellement d'autres ontologies périphériques, enfin des modules d'alignement importent au moins une ontologie du réseau et une ontologie externe.

Comme illustré sur la figure 2, la suite d'ontologies ETSI SAREF est composée d'ontologies définissant des patrons génériques comme SAREF4SYST [21], d'une ontologie noyau SAREF Core [23] illustrée dans la figure 1, et de différentes extensions développées pour des domaines verticaux distincts : SAREF4ENER pour l'energie [12], SAREF4ENVI pour l'environement [14], SAREF4BLDG pour les bâtiments intelligents [14], SAREF4CITY pour les villes intelligentes [15], SAREF4INMA pour les l'industrie manufacturière [16], SAREF4AGRI pour l'agriculture [17], SAREF4AUTO pour l'automobile [18], SAREF4EHAW pour la e-santé et le bon vieillissement [19], SAREF4WEAR pour les wearables [20], SAREF4WATR pour la gestion de l'eau [13], SAREF4LIFT pour les ascenseurs intelligents [24].

Deux choix de conception sont récurrents dans ces réseaux d'ontologies : (1) la définition des espaces de noms pour chaque module, et (2) le choix du module dans lequel un terme est défini. Choisir un espace de nom distinct pour chaque module permet d'identifier facilement de quel module un terme est issu. Cela simplifie également la publication des ontologies dans le respect de la bonne pratique qui consiste à rendre une description de chaque terme accessible à son IRI (des IRI de type hash '#' peuvent être utilisées). Cependant, cette approche pose trois problèmes :

Premièrement, il est parfois difficile en tant qu'utilisateur de ces ontologies, de se souvenir quel est l'espace de nom pour chaque concept. Nous avons par exemple une variété de sous-classes de saref:Property réparties dans les espaces de noms des différentes extensions, selon là où on a eu besoin de les définir en premier : saref:Temperature, saref:Humidity, saref:Power, s4ener:Power, s4ener:PowerMax, s4ener:PowerStandardDeviation, s4inma:Size, saref:Light, s4envi:LightProperty, ainsi que les instances de saref:Property suivantes : s4envi:Frequency, s4wear:SoundLevel, s4wear:BatteryRemainingTime, s4watr:Conductivity, s4wear:Temperature. Une approche alternative aurait consisté à utiliser un espace de nom unique et des IRIs de type slash '/', et implémenter des redirections de l'IRI de chaque terme vers le document qui décrit l'ontologie où il est défini.

Deuxièmement, l'expérience montre qu'il peut être pertinent de déplacer un terme d'un module vers un autre. Par exemple SAREF4CITY V1.1.1 a introduit le concept de s4city:FeatureOfInterest, et il a été décidé lors du développement de SAREF Core V3.1.1 que ce concept devait être déplacé dans l'ontologie noyau. Il est donc maintenant identifié par saref:FeatureOfInterest, et les implémentations de SAREF4CITY ont dû être modifiées. Ce problème ne se serait pas posé si une approche basée sur un espace de nom unique et des IRIs de type slash avait été adoptée.

Enfin, ce que montre la liste des classes et instances de saref:Property, c'est qu'au sein même de la communauté des développeurs de SAREF, des choix de modélisation et de nommage sont parfois variés. Il nous apparaîtrait donc important de re-baser le développement de SAREF sur des patrons d'ontologies pour harmoniser son développement.

5 Patrons

En ingénierie logicielle, un patron est défini comme une *spécification abstraite d'une composition d'objets qui fait que toute instance de la composition possède une propriété donnée.* [37]. Le concept a été transposé à l'ingénierie des ontologies avec les *Ontology Design Patterns* [27, 6, 28]. C'est le sujet par exemple de l'état de l'art [7].

Un des livrables du projet STF 556 de l'ETSI est le rapport technique TR 103 549 nommé "Consolidation de SAREF et de sa communauté d'utilisateurs industriels, sur la base de l'expérience du projet EUREKA ITEA 12004 SEAS". Ce rapport identifie les patrons implicitement existant dans SAREF, et qu'il pourrait convenir de formaliser pour aboutir à une version consolidée de l'ontologie SAREF [11]. Par exemple dans la version V2.1.1 de SAREF Core, les fonctions de détection, d'actionnement et de mesure sont des types de fonctions. Habituellement, une fonction (par exemple saref:StartStopFunction) a une ou plusieurs commandes pour la déclencher (par exemple, pour saref:StartStopFuncton, ce devrait être soit une saref:StartCommand soit une saref:StopCommand). Certaines commandes agissent sur certains états (saref:StartStopCommand agit sur un certain saref:StartStopState). Il conviendrait par exemple de s'assurer que toutes les sous-classes de la classe saref:Command soient décrites de la même manière. Par exemple, des sous-classes de saref:Command avaient des instances génériques, associées à aucune réelle action. SAREF avait aussi une commande nommée saref:PauseCommand, qui n'était associée à aucune fonction.

Des patrons peuvent être instanciés avec les éléments pris dans un ou plusieurs dimensions orthogonales. Par exemple, SAREF4ENER définit s4ener:EnergyMax, s4ener:EnergyMin, s4ener:EnergyExpected, s4ener:EnergyStandardDeviation, s4ener:PowerMax, s4ener:PowerMin, s4ener:PowerExpected, s4ener:PowerStandardDeviation. Gérer manuellement l'ajout par exemple d'un

nouveau type d'agrégat *Average* implique de créer de nombreuses propriétés, comme s4ener:EnergyAverage, s4ener:PowerAverage.
Une solution partielle à ce problème consiste à découpler les dimensions. Dans l'exemple ci-dessus : le type de propriété, et le type d'agrégat. Dans les ontologies SEAS [45] nous avons proposé une modélisation qui vise à éviter ces situations, en découplant les dimensions. Une entité d'intérêt est liée à une seule instance de la classe seas:Property par le biais d'une relation qui serait nommée seas:hasElectricConsumption par exemple. Cette instance de propriété peut alors être d'un type générique seas:PowerProperty, et des évaluations de cette propriété peuvent être définies et multi-typées. Par exemple seas:Evaluation, seas:MinEvaluation, seas:AverageEvaluation, seas:SumEvaluation.
De plus, vouloir modifier généralement comment sont décrites les sous-classes d'une classe principale comme saref:Property peut s'avérer fastidieux, car chaque instance du patron doit être revue manuellement. Parmi les travaux récents qui permettent d'automatiser la génération des ontologies à partir de patrons et de description des instances, on peut citer le système *Reasonable Ontology Template* OTTR [1] [60], *Generic Ontology Design Patterns* [43, 44], ou *Dead Simple OWL Design Patterns* (DOS-DPs) [52]. OTTR permet de déclarer des patrons d'ontologie, et de générer automatiquement des instances de ces patrons à partir d'un document externe, par exemple un document Excel écrit par les experts de domaine.
L'ontologie SAREF4SYST [21], illustrée sur la figure 3 et inspirée de SEAS, est la première ontologie de patron incorporée à SAREF. Elle définit un modèle d'ontologie qui peut être instancié pour différents domaines. SAREF4SYST définit les systèmes, les connexions entre les systèmes et les points de connexion auxquels les systèmes peuvent être connectés. Ces concepts de base peuvent être utilisés de manière générique pour définir la topologie des entités d'intérêt, et peuvent être spécialisés pour de multiples domaines. Par exemple, pour décrire des zones à l'intérieur d'un bâtiment (systèmes), qui partagent une frontière (connexions). Les propriétés des systèmes sont généralement des variables d'état (par exemple, la population des agents, la température), tandis que les propriétés des connexions sont généralement des flux (par exemple, le flux de chaleur). SAREF4SYST a deux objectifs principaux : d'une part, étendre SAREF avec la capacité de représenter la topologie générale des systèmes et comment ils sont connectés ou interagissent et, d'autre part, illustrer comment les patrons d'ontologie peuvent aider à assurer une structure homogène de l'ontologie SAREF globale et accélérer le développement d'extensions.

1. https://ottr.xyz/

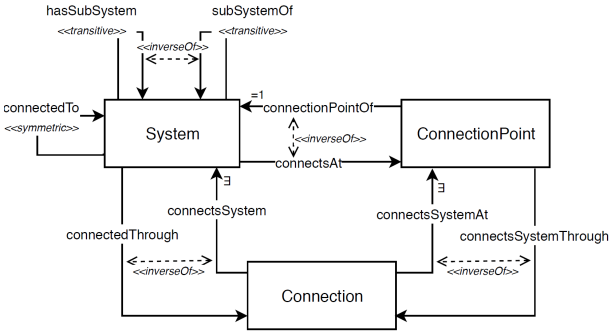

FIGURE 3 – Aperçu du patron d'ontologie SAREF4SYST (Source : [21])

6 Environnement de développement

Différents logiciels spécialisés existent pour l'édition des ontologies [2], Stanford Protégé [49] étant probablement le plus connu et utilisé. Des logiciels professionnels existent comme TopBraid Enterprise Vocabulary Net [2] par exemple, qui est une extension de l'environnement de développement intégré Apache Eclipse.

Pour l'édition du code source avec la syntaxe Turtle 1.1 par exemple, des plugins commencent à fleurir pour la plupart des éditeurs de texte multi-langage. Par exemple le plugin *Linked Data syntaxes* [3] pour Sublime Text [4], ou l'extension

2. https://www.topquadrant.com/the-topbraid-evn-ontology-editor/
3. https://github.com/blake-regalia/linked-data.syntaxes
4. https://www.sublimetext.com/

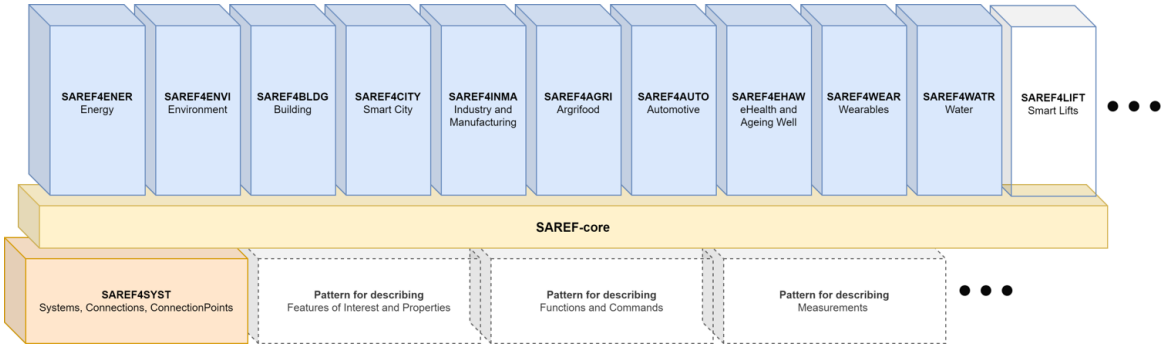

FIGURE 2 – L'ontologie SAREF et ses différents modules

Stardog RDF Grammars [5] pour Visual Studio Code [6]. Ces extensions permettent la coloration syntaxique des fichiers RDF, et ainsi d'identifier rapidement des erreurs de syntaxe. Des fonctionnalités supplémentaires peuvent être attendues d'un environnement de développement intégré d'ontologies. Nous avons *forké* par exemple le plugin *Linked Data syntaxes* pour implémenter l'exécution de règles SPARQL-Generate [46] avec la combinaison de touche CTRL+B [7]. Un récent projet de nos étudiants [8] consistait à implémenter une fonctionnalité de navigation dans des projets contenant de nombreux fichiers RDF : un contrôle-clic sur un terme RDF permet de naviger vers là où est défini le terme en question : soit un fichier local, soit une URL dans le navigateur. Lorsque l'on écrit une IRI préfixée, le système peut également vérifier que le préfixe est déclaré, et éventuellement insérer automatiquement le nouveau préfixe dans l'entête si il est connu de la base de données `prefix.cc` [9].

Le *Linting*, dont le nom vient d'une commande UNIX de pré-processeur pour le langage C, est une approche qui consiste à analyser statiquement le code source d'un logiciel pour détecter des erreurs, bugs, ou erreurs de style. Développer un *linter* pour supporter le processus d'édition des ontologies permettrait de limiter la difficulté d'éditer des ontologies de qualité. Par exemple avec Jena Eyeball, [10], ou RDFLint [11]. Ce dernier est intégré dans l'extension *RDF language support via rdflint* [12] de Visual Studio Code, et permet entre autre d'exécuter des requêtes SPARQL, de valider des contraintes SHACL, ou de valider que les littéraux sont bien formés. Bien que le développement de tels outils soit difficile à valoriser d'un point de vue de la recherche, ils nous semblent extrêmement importants pour contribuer à abaisser le niveau de compétence nécessaire pour développer des ontologies d'une bonne qualité.

7 Nommage des versions

Les ontologie sont amenées à évoluer, pendant le travail de développement potentiellement collaboratif, mais également après une première publication si des évolutions sont nécessaires. Différentes pratiques de nommage des versions existent pour les logiciels, les plus connus étant le *Semantic Versioning* [13] et le *Calendar Versioning* [14]. Une transposition de semver aux ontologies a été proposée [65], et est assez communément utilisé aujourd'hui.

Le langage OWL définit deux types d'identifiants pour les ontologies : un identifiant de série d'ontologie (l'instance de owl:Ontology), et l'identifiant de version d'ontologie (l'objet de la métadonnée owl:versionIRI) [48, Sec. 3.3]. Les utilisateurs peuvent alors choisir d'importer une ontologie par son identifiant de série et ainsi suivre les évolutions de l'ontologie, ou son identifiant de version et s'assurer que rien ne cassera en cas d'évolution non rétrocompatible.

Une erreur de conception commune même à certaines ontologies du W3C consiste à inclure l'identifiant de version, ou la date de publication dans l'identifiant de série d'ontologie. Par exemple, QUDT V1.1 avait l'identifiant `http://qudt.org/1.1/schema/qudt#` (corrigé depuis la version 2). Les identifiants de RDF, RDFS, et OWL, contiennent respectivement `1999/02/22`, `2000/01`, et `2002/07`. Il n'est dans ce cas possible de conserver l'identifiant pour la version suivante de l'ontologie sans créer une incohérence. RDFS 1.1 publié en 2014 conserve l'année 2000 dans son identifiant.

L'évolution d'une ontologie peut avoir différents impacts sur les artefacts (ontologies, bases de connaissances, logiciels) qui l'utilisent. Dans sa thèse, Omar Alqawasmeh [5, 57] étudie ces différents problèmes et propose des contremesures. Une recommandation adoptée dans le travail de développement des ontologies SAREF est par exemple de s'assurer que l'on importe des ontologies par leur IRI de version, et non pas par l'identifiant de série d'ontologie. Par exemple, SAREF4LIFT V1.1.1 importe SAREF Core V3.1.1, SAREF4SYST V1.1.2, et SAREF4BLDG V1.1.2.

Avec SAREF, nous allons plus loin dans l'adoption du *Semantic Versioning*. Chaque module de l'ontologie possède une version distincte, composée de trois numéros : un *MAJOR*, un *Minor*, et un *patch*. L'incrémentation du *MAJOR* indique une coupure de la rétrocompatibilité. L'incrémentation du *Minor* indique l'ajout de fonctionnalités. L'incrémentation du *patch* indique la correction d'un bug. On pourrait donc importer un module SAREF, par exemple SAREF Core, avec différentes IRI, exprimant différents choix : 1. `https://saref.etsi.org/core/` redirigera vers la dernière version Vx.y.z 2. `https://saref.etsi.org/core/v3` redirigera vers la dernière version V3.y.z 3. `https://saref.etsi.org/core/v3.1` redirigera vers la dernière version V3.1.z . On pourrait également proposer un schéma général d'IRI, s'appuyant sur l'expression d'une spécification de version comme le *Maven Dependency Version Range* [15] pour Java, ou PEP440 [16] pour Python.

8 Contrôle des versions et workflow d'édition

Le logiciel de gestion de versions distribuées git s'est rapidement imposé dans l'édition collaborative de logiciels, et les plateformes de type GitHub ou Gitlab ont démocratisé différents flux opérationnels (*workflows*) basés sur

5. `https://marketplace.visualstudio.com/items?itemName=stardog-union.stardog-rdf-grammars`
6. `https://visualstudio.microsoft.com/`
7. `https://w3id.org/sparql-generate/sublime.html`
8. `https://github.com/clement000/linked-data.syntaxes`
9. `https://prefix.cc/`
10. `https://jena.apache.org/documentation/archive/eyeball/eyeball-manual.html`
11. `https://github.com/imas/rdflint`
12. `https://marketplace.visualstudio.com/items?itemName=takemikami.vscode-rdflint`
13. `https://semver.org`
14. `https://calver.org`
15. `https://cwiki.apache.org/confluence/display/MAVENOLD/Dependency+Mediation+and+Conflict+Resolution`
16. `https://peps.python.org/pep-0440/#version-specifiers`

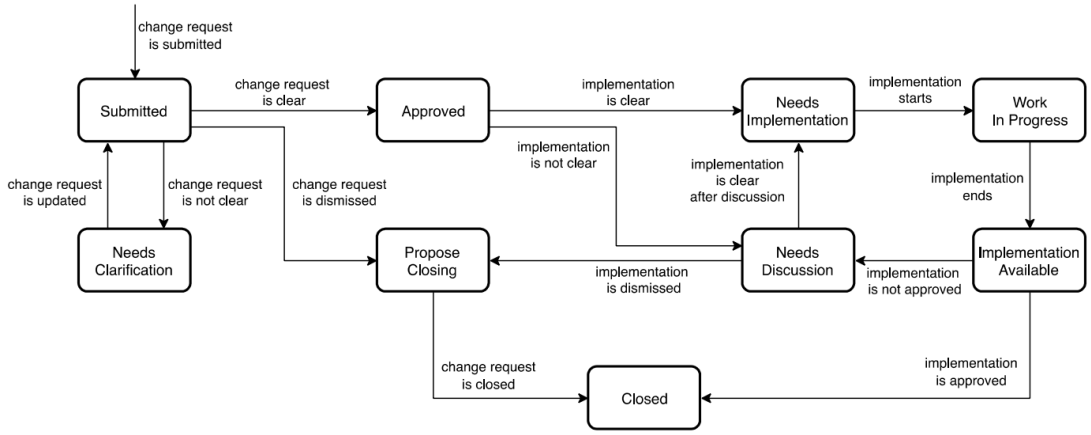

FIGURE 4 – Exemple du flux opérationnel pour l'édition d'une version de SAREF (Source : [22])

les branches (fonctionnalité offerte par git), les forks, tickets, jalons, et requêtes de fusion (*pull request* pour Github, *merge request* pour Gitlab).

Des retours d'expérience de workflows sur Github et identification des meilleures pratiques commencent à être publiés [8, 3]. Pour l'édition des ontologies SAREF, nous utilisons le portail publique ETSI Forge `https://saref.etsi.org/sources/`, et avons publié une spécification technique établissant différents flots de travail pour (1) la création d'une version d'ontologie, (2) le développement d'une version d'ontologie, (3) la publication d'un projet [22, Clauses 6.1, 7.1, 8.1]. Nous utilisons quatre type de branches : des branches `issue-x` pour le travail sur un ticket, des branches `develop-vx.y.z` pour le travail sur une version, des branches `prerelease-vx.y.z` pour le travail de validation final de l'ontologie, et des branches `release-vx.y.z` pour les versions publiées. Des règles de protection sont définies pour interdire aux développeurs d'une ontologie de pousser directement des changements sur les branches `develop-vx.y.z`, ou d'accepter directement des *merge request* sur les branche `prerelease-vx.y.z`. Les tickets sont disponibles à l'adresse `https://labs.etsi.org/rep/groups/saref/-/issues`.

Agec git, une version d'un logiciel, nommé *commit*, contient zéro ou plusieurs *commit* parent, la liste des fichiers modifiés dans ce commit, les versions compressées des nouvelles versions de ces fichiers, et d'autres métadonnées comme la date et l'identifiant de l'auteur du *commit*. Instaurer l'état d'un logiciel pour un commit donné consiste donc à parcourir l'arbre de ses ancêtres, et décompresser pour chaque commit ancêtre les fichiers qui n'ont pas été modifiés ultérieurement.

Git peut se baser sur différents algorithmes de calcul des différences entre fichiers texte (*diff*), défini par la variable `diff.algorithm`. Pour l'édition des ontologies, ceci pose un problème car les éditeurs spécialisés peuvent complètement transformer la sérialisation d'une ontologie, puisque chacun se base sur une librairie de sérialisation différente (Apache RIOT pour Protégé, RDF4J RIO pour Topbraid-Composer). En pratique, cela rend très difficile l'évaluation des changements implémentés lorsqu'il faut valider une Pull Request, puisque tout semble avoir été changé.

Il serait théoriquement possible de modifier l'algorithme qu'utilise git pour la détection de différences entre deux versions, par exemple pour Promptdiff [51], Ecco [30], ou OWLDiff [42]. Cependant il faudrait que le serveur GitHub ou GitLab puisse utiliser le même algorithme pour visualiser le résultat du *diff*. Une approche alternative proposée par exemple dans [33] consiste à utiliser les crochets *hooks* [17], qui s'assurent que le même outil de serialisation est utilisé avant chaque commit.

Deux pratiques principales existent pour identifier des versions de logiciels avec git : les étiquette (*tag*) de version, et les branches de sortie (*release branch*). Le développement des ontologies SAREF utilise cette deuxième approche, qui permet de continuer à faire évoluer la documentation ou les exemples même lorsque l'on fige une ontologie.

9 Automatisation

L'automatisation est un sujet majeur en développement logiciel et a pour objectif d'accélérer la production et la qualité des logiciels, éviter les tâches redondantes, et limiter les mauvaises versions de logiciels. Différentes tâches peuvent être automatisées en ingénierie des ontologies, par exemple avec les outils de *linting* présentés dans la section 6, ou l'interface en ligne de commande disponible dans des frameworks comme Apache Jena [18]. L'outil ROBOT développé par la communauté OBO [39] permet d'exécuter automatiquement un certain nombre de tâches pour convertir, raisonner, importer, extraire des modules, filtrer des axiomes, requêter ou vérifier la bonne exécution de requêtes SPARQL pour évaluer des tests unitaires, générer un rapport d'erreurs, réparer si possible, instancier des patrons, et assembler ces tâches dans des workflows.

Pour les projets de développement d'ontologies qui utilisent

17. `https://git-scm.com/book/fr/v2/Personnalisation-de-Git-Crochets-Git`
18. `https://jena.apache.org/documentation/tools/index.html`

git, il est possible d'extraire automatiquement des informations à injecter dans les métadonnées de l'ontologie. Par exemple les trois commandes suivantes trouvent pour une ontologie `onto.ttl` : 1. la date de premier commit, ce qui peut permettre de renseigner la propriété dct:created, 2. la date de dernière modification (propriété dct:modified), 3. les auteurs classés par ordre décroissant de nombre de modifications (propriété owl:contributor).

```
git log --diff-filter=A --format='%ad' --date=short -- onto.ttl
git log -1 --format='%ad' --date=short -- onto.ttl
git log -- onto.ttl | grep Author | sort | uniq -c | sort -n -r
```

Dans le projet STF 578, nous avons spécifié un ensemble de règles auxquelles un dépôt d'ontologie SAREF doit se conformer dans la spécification technique ETSI TS 103 673 [22, Clause 9], et avons développé l'application SAREF Pipeline qui permet d'évaluer chacune de ces règles avec un niveau d'exigence. voici une liste non exhaustive des points évalués : (a) Structure du répertoire de dépôt, (b) présence d'un fichier de licence défini, (c) spécification des besoins de l'ontologie, (d) présence d'un fichier `/saref4[a-z]{4}.ttl/` bien formé, (e) déclaration de préfixes conformes, (f) présence d'une déclaration d'ontologie, avec une IRI de série et une IRI de version conformes au nommage de la branche git (ex : `develop-v2.1.1`, (g) imports éventuels d'autres ontologies SAREF par leur IRI de version, (h) présence des créateurs et contributeurs, (i) convention de nommage pour les classes, propriétés, instances, (j) présence de métadonnées pour les termes, (k) l'ontologie doit être OWL 2 DL, (l) l'ontologie doit être consistante, (m) chaque classe doit être satisfiable (n) aucun pitfall détecté par OOPS ! [56], (o) présence de tests, (p) présence et qualité des exemples, (q) existence des termes utilisés.

Certains de ces tests utilisent des shapes SHACL [41], d'autres les fonctionnalités de OWLAPI après avoir cloné les dépôts nécessaires. Le dossier des messages de l'application donne une vue globale de toutes les erreurs qui peuvent être identifiées [19]. Cette application peut être utilisée avec une interface graphique (figure 5) ou en ligne de commande (figure 6). Le rapport d'erreur est formatté en markdown, ce qui permet d'ouvrir rapidement un ticket pour traiter le problème collaborativement (figure 7). Finalement, l'application génère différentes sérialisations pour les ontologies et les exemples, et une documentation HTML inspirée de LODE et réécrite avec SPARQL-Generate [46]. Voir par exemple `https://saref.etsi.org/core` ou `https://saref.etsi.org/core/Command`.

Cette application est monolithique et peut difficilement être réutilisée pour d'autres projets d'ingénierie d'ontologie. Nous travaillons sur des améliorations pour les tâches d'ingénierie des ontologies pour les projets ANR HyperAgents (ANR-19-CE23-0030-01) et ANR CoSWoT (ANR-19-CE23-0012-04).

19. `https://labs.etsi.org/rep/saref/saref-pipeline/-/tree/master/src/main/resources/messages`

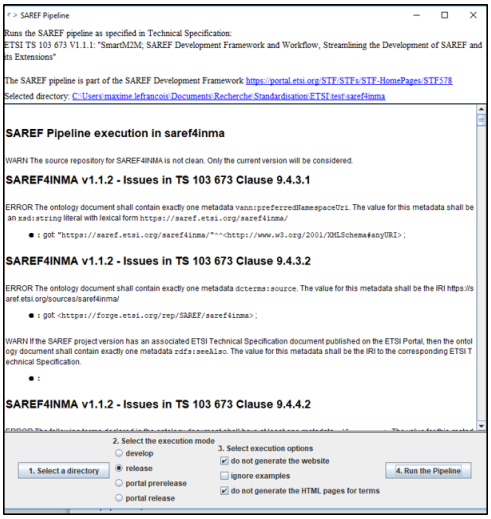

FIGURE 5 – Exécution du pipeline SAREF avec l'interface graphique `https://saref.etsi.org/sources/saref-pipeline/`

```
saref-pipeline/target$ java -jar saref-pipeline.jar
usage: java -jar saref-pipeline.jar <mode> [<options>] [<target>]

Runs the SAREF pipeline as specified in Technical Specification:
  ETSI TS 103 673 V1.1.1: "SmartM2M; SAREF Development Framework and
            Workflow, Streamlining the Development of SAREF and its
            Extensions"

<mode> can take the following values:
  develop    Run the SAREF pipeline in relax mode in the target SAREF
             project
  release    Run the SAREF pipeline in strict mode in the target SAREF
             project
  prerelease-portal    Operate a strict check and generate the portal for
             pre-release or release branches of each source in the
             configuration file `.saref-repositories.yml`
  release-portal    Operate a strict check generate the portal for release
             branches of each source in the configuration file
             `.saref-repositories.yml`
  clean      Remove all files generated by the previous executions
  help       Displays this message

<target> points to the target directory. By default, target is the current
             directory.

<options> can take the following values:
 -e,--no-examples   Do not check examples
 -s,--no-site       Do not generate the static portal
 -t,--no-terms      Do not generate the static portal for terms
```

FIGURE 6 – Exécution du pipeline SAREF en ligne de commande `https://saref.etsi.org/sources/saref-pipeline/`

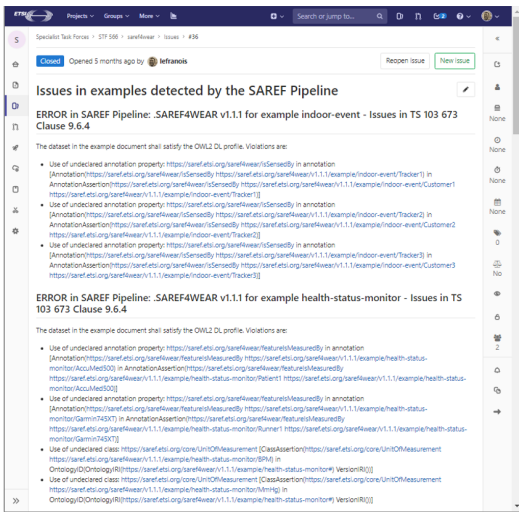

FIGURE 7 – La sortie du pipeline SAREF est formatée en markdown et peut être utilisée pour créer un ticket

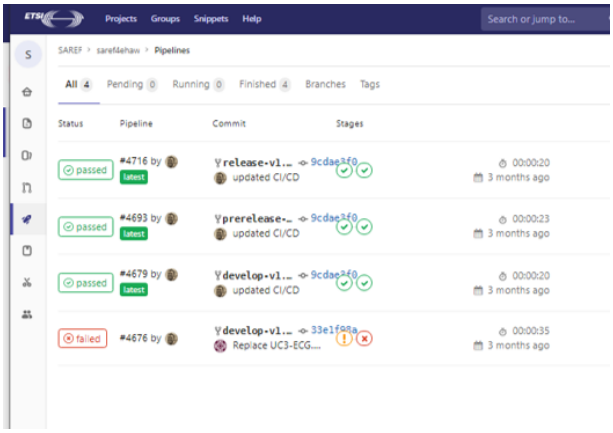

FIGURE 8 – Aperçu des pipeline d'intégration et de déploiement continu : *Snapshot*, *Staging*, *Manual release*. Source : `https://saref.etsi.org/sources/saref4ehaw/-/pipelines`

10 Intégration et deploiement continu

A l'instar des méthodes Agile qui visent à améliorer les collaborations entre les clients d'un projet logiciel et les développeurs, les méthodes DevOps améliorent les collaborations entre les développeurs et les professionnels des opérations informatiques. Jenkins [20], Travis CI [21], Circle CI [22], Gitlab CI/CD [23], Github Actions [24], sont tous des frameworks qui permettent de spécifier des pipelines de tâches qui seront exécutés automatiquement lorsque par exemple un commit est poussé sur le serveur. Avant la démocratisation de ces frameworks, quelques approches préliminaires ont été proposées dans la communauté de l'ingénierie des ontologies en utilisant les applications Github [25]. Par exemple VoCol [34] ou OnToology [4]. *Ontology Development Kit* (ODK) [47] utilise Travis CI pour exécuter des workflows avec ROBOT.
Dans le projet STF 578, nous avons configuré Gitlab CI/CD dans chaque dépôt des ontologies SAREF, pour qu'il exécute le pipeline SAREF différemment selon le type de branche où est poussé un commit (issue, develop, prerelease, release), et pousse finalement automatiquement les fichiers de sortie vers le portail de documentation de SAREF `https://saref.etsi.org/`. La figure 8 illustre l'exécution automatique des pipelines SAREF.

11 Conclusion

Dans cet article nous avons montré que les méthodologies et techniques de développement logiciel ont eu des répercussions importantes en ingénierie de l'ontologie, au moins pour les neuf thématiques identifiées. Nous avons illustré comment le framework de développement et de publication de l'ontologie ETSI SAREF a été spécifié pour tirer partie des dernières méthodologies et techniques disponibles. Nous appliquons et améliorons actuellement ces travaux dans les tâches d'ingénierie des ontologies pour les projets ANR HyperAgents (ANR-19-CE23-0030-01) et ANR CoSWoT (ANR-19-CE23-0012-04)

20. `https://www.jenkins.io/`
21. `https://travis-ci.org/`
22. `https://circleci.com/`
23. `https://docs.gitlab.com/ee/ci/`
24. `https://github.com/features/actions`
25. `https://docs.github.com/en/developers/apps`

Références

[1] Abdelghany Salah Abdelghany, Nagy Ramadan Darwish, and Hesham Ahmed Hefni. An agile methodology for ontology development. *International Journal of Intelligent Engineering and Systems*, 12(2) :170–181, 2019.

[2] Emhimed Salem Alatrish. Comparison of ontology editors. *eRAF Journal on Computing*, 4 :23–38, 2012.

[3] Dean Allemang, Pawel Garbacz, Przemyslaw Gradzki, Elisa Kendall, and Robert 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.

[4] Ahmad Alobaid, Daniel Garijo, María Poveda-Villalón, Idafen Santana-Perez, Alba Fernández-Izquierdo, and Oscar Corcho. Automating ontology engineering support activities with ontoology. *Journal of Web Semantics*, 57 :100472, 2019.

[5] Omar Alqawasmeh. *Towards a collaborative framework for ontology engineering : Impact on ontology evolution and pitfalls in ontology networks and versioned ontologies*. Theses, Université de Lyon, September 2020.

[6] Eva Blomqvist and Kurt Sandkuhl. Patterns in ontology engineering : Classification of ontology patterns. In *International Conference on Enterprise Information System*, pages 413–416, 2005.

[7] Giuseppe Cota, Marlena Daquino, and Gian Luca Pozzato. *Applications and Practices in Ontology Design, Extraction, and Reasoning*, volume 49. IOS Press, 2020.

[8] Robert Crystal-Ornelas, Charuleka Varadharajan, Ben Bond-Lamberty, Kristin Boye, Madison Burrus, Shreyas Cholia, Michael Crow, Joan Damerow, Ranjeet Devarakonda, Kim S Ely, et al. A guide to using github for developing and versioning data standards and reporting formats. *Earth and Space Science*, 8(8), 2021.

[9] Joel Cummings and Deborah Stacey. Lean ontology development : An ontology development paradigm based on continuous innovation. In *Knowledge Engineering and Ontology Development*, pages 365–372, 2018.

[10] Antonio De Nicola and Michele Missikoff. A lightweight methodology for rapid ontology engineering. *Communications of the ACM*, 59(3) :79–86, 2016.

[11] ETSI. SmartM2M ; Guidelines for consolidating SAREF with new reference ontology patterns, based on the experience from the ITEA SEAS project. ETSI Technical Report 103 549 V1.1.1., 07 2019.

[12] ETSI. SmartM2M ; Extension to SAREF ; Part 1 : Energy Domain. ETSI Technical Specification 103 410-1 V1.1.2., 05 2020.

[13] ETSI. SmartM2M ; Extension to SAREF ; Part 10 : Water Domain. ETSI Technical Specification 103 410-10 V1.1.1., 07 2020.

[14] ETSI. SmartM2M ; Extension to SAREF ; Part 2 : Environment Domain. ETSI Technical Specification 103 410-2 V1.1.2., 05 2020.

[15] ETSI. SmartM2M ; Extension to SAREF ; Part 4 : Smart Cities Domain. ETSI Technical Specification 103 410-4 V1.1.2., 05 2020.

[16] ETSI. SmartM2M ; Extension to SAREF ; Part 5 : Industry and Manufacturing Domain. ETSI Technical Specification 103 410-5 V1.1.2., 05 2020.

[17] ETSI. SmartM2M ; Extension to SAREF ; Part 6 : Smart Agriculture and Food Chain Domains. ETSI Technical Specification 103 410-6 V1.1.2., 05 2020.

[18] ETSI. SmartM2M ; Extension to SAREF ; Part 7 : Automotive Domain. ETSI Technical Specification 103 410-7 V1.1.1., 07 2020.

[19] ETSI. SmartM2M ; Extension to SAREF ; Part 8 : eHealth/Ageing-well Domain. ETSI Technical Specification 103 410-8 V1.1.1., 07 2020.

[20] ETSI. SmartM2M ; Extension to SAREF ; Part 9 : Wearables Domain. ETSI Technical Specification 103 410-9 V1.1.1., 07 2020.

[21] ETSI. SmartM2M ; SAREF consolidation with new reference ontology patterns, based on the experience from the SEAS project. ETSI Technical Specification 103 548 V1.1.2., 06 2020.

[22] ETSI. SmartM2M ; SAREF Development Framework and Workflow, Streamlining the Development of SAREF and its Extensions. ETSI Technical Specification 103 673 V1.1.1., 2020.

[23] ETSI. SmartM2M ; Smart Applications ; Reference Ontology and oneM2M Mapping. ETSI Technical Specification 103 264 V3.1.1., 02 2020.

[24] ETSI. SmartM2M ; Extension to SAREF ; Part 11 : Lift Domain. ETSI Technical Specification 103 410-11 V1.1.1., 07 2021.

[25] Alba Fernández-Izquierdo and Raúl García-Castro. Themis : a tool for validating ontologies through requirements. In *Software Engineering and Knowledge Engineering*, pages 573–753, 2019.

[26] Mariano Fernández-López, Asunción Gómez-Pérez, and Natalia Juristo. Methontology : from ontological art towards ontological engineering. 1997.

[27] Aldo Gangemi. Ontology design patterns for semantic web content. In *International semantic web conference*, pages 262–276. Springer, 2005.

[28] Aldo Gangemi and Valentina Presutti. Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer, 2009.

[29] Asunción Gómez-Pérez and Mari Carmen 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.

[30] Rafael S Gonçalves, Bijan Parsia, and Ulrike Sattler. Ecco : A hybrid diff tool for owl 2 ontologies. In *OWLED*, volume 849. Citeseer, 2012.

[31] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Modularity and web ontologies. In *Knowledge Representation*, pages 198–209, 2006.

[32] Michael Grüninger and Mark S Fox. Methodology for the design and evaluation of ontologies. 1995.

[33] Lavdim Halilaj, Irlán Grangel-González, Maria-Esther Vidal, Steffen Lohmann, and Sören Auer. Proactive prevention of false-positive conflicts in distributed ontology development. In *Knowledge Engineering and Ontology Development*, pages 43–51, 2016.

[34] Lavdim Halilaj, Niklas Petersen, Irlán Grangel-González, Christoph Lange, Sören Auer, Gökhan Coskun, and Steffen Lohmann. Vocol : An integrated environment to support version-controlled vocabulary development. In *European Knowledge Acquisition Workshop*, pages 303–319. Springer, 2016.

[35] Maia Hristozova and Leon 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.

[36] IEEE. *Guide to the Software engineering body of knowledge v3.0*. IEEE Computer society, 2014.

[37] ISO/IEC/IEEE. Systems and software engineering — vocabulary. Standard ISO/IEC/IEEE 24765:2017, 2017.

[38] Alba Fernández Izquierdo. Ontology verification based on lexico-syntactic patterns, November 2020.

[39] Rebecca C Jackson, James P Balhoff, Eric Douglass, Nomi L Harris, Christopher J Mungall, and James A Overton. Robot : a tool for automating ontology workflows. *BMC bioinformatics*, 20(1) :1–10, 2019.

[40] Haruhiko Kaiya and Motoshi Saeki. Using domain ontology as domain knowledge for requirements elicitation. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 189–198. IEEE, 2006.

[41] Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). W3C Recommendation, W3C, July 20 2017.

[42] Petr Kremen, Marek Smid, and Zdenek Kouba. Owldiff : A practical tool for comparison and merge of owl ontologies. In *2011 22nd International Workshop on Database and Expert Systems Applications*, pages 229–233. IEEE, 2011.

[43] Bernd Krieg-Brückner and Till Mossakowski. Generic ontologies and generic ontology design patterns. In *WOP@ ISWC*, 2017.

[44] Bernd Krieg-Brückner, Till Mossakowski, and Mihai Codescu. Generic ontology design patterns : Roles and change over time. *Advances in Pattern-Based Ontology Engineering*, 51 :25, 2021.

[45] Maxime Lefrançois. Planned ETSI SAREF extensions based on the W3C&OGC SOSA/SSN-compatible SEAS ontology paaerns. In *Workshop on semantic interoperability and standardization in the IoT, SIS-IoT*, page 11p, 2017.

[46] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *European Semantic Web Conference*, pages 35–50. Springer, 2017.

[47] Nicolas Matentzoglu, Chris Mungall, and Damien Goutte-Gattat. Ontology development kit, July 2021. If you use this software, please cite it as below.

[48] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. Owl 2 web ontology language : Structural specification and functional-style syntax. W3c recommendation, W3C, 2009.

[49] Mark A Musen. The protégé project : a look back and a look forward. *AI matters*, 1(4) :4–12, 2015.

[50] Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101 : A guide to creating your first ontology, 2001.

[51] Natalya Fridman Noy, Mark A Musen, et al. Promptdiff : A fixed-point algorithm for comparing ontology versions. *AAAI/IAAI*, 2002 :744–750, 2002.

[52] David Osumi-Sutherland, Melanie Courtot, James P Balhoff, and Christopher Mungall. Dead simple owl design patterns. *Journal of biomedical semantics*, 8(1) :1–7, 2017.

[53] Silvio 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)*, pages 1–14, 2016.

[54] Helena Sofia Pinto, Steffen Staab, and Christoph Tempich. Diligent : Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In *ECAI*, volume 16, page 393. Citeseer, 2004.

[55] María Poveda-Villalón, Alba Fernández-Izquierdo, Mariano Fernández-López, and Raúl García-Castro. Lot : An industrial oriented ontology engineering framework. *Engineering Applications of Artificial Intelligence*, 111 :104755, 2022.

[56] María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. Oops !(ontology pitfall scanner !) : An on-line tool for ontology evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2) :7–34, 2014.

[57] Omar Qawasmeh, Maxime Lefrançois, Antoine Zimmermann, and Pierre Maret. Pitfalls in networked and versioned ontologies. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management - 11th International Joint Conference, IC3K 2019, Vienna, Austria, September 17-19, 2019, Revised Selected Papers*, volume 1297 of *Communications in Computer and Information Science*, pages 185–212. Springer, 2019.

[58] Lila Rao, Han Reichgelt, and Kweku-Muata Osei-Bryson. Knowledge elicitation techniques for deriving competency questions for ontologies. In *International Conference on Enterprise Information System*, pages 105–110, 2008.

[59] Amir Azim Sharifloo and Mehrnoush Shamsfard. Using agility in ontology construction. In *Formal Ontologies Meet Industry*, volume 174 of *Frontiers in Artificial Intelligence and Applications*, pages 109–119. IOS Press, 2008.

[60] Martin G Skjæveland, Henrik Forssell, Johan W Klüwer, Daniel Lupp, Evgenij Thorstensen, and Arild Waaler. Pattern-based ontology design and instantiation with reasonable ontology templates. *A Higher-Level View of Ontological Modeling*, page 69, 2019.

[61] Steffen Staab, Rudi Studer, H-P Schnurr, and York Sure. Knowledge processes and ontologies. *IEEE Intelligent systems*, 16(1) :26–34, 2001.

[62] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Boris Villazón-Terrazas. How to write and use the ontology requirements specification document. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 966–982. Springer, 2009.

[63] Akkharawoot Takhom, Sasiporn Usanavasin, Thepchai Supnithi, and Prachya Boonkwan. A collaborative framework supporting ontology development based on agile and scrum model. *IEICE TRANSACTIONS on Information and Systems*, 103(12) :2568–2577, 2020.

[64] Mike Uschold and Michael Gruninger. Ontologies : Principles, methods and applications. *The knowledge engineering review*, 11(2) :93–136, 1996.

[65] Max Volkel, Wolf Winkler, York Sure, S Ryszard Kruk, and Marcin Synak. Semversion : A versioning system for rdf and ontologies. In *Proc. of ESWC*, 2005.