Data & Knowledge Engineering 133 (2021) 101889

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

An ontology knowledge inspection methodology for quality assessment and continuous improvement

Gabriela R. Roldán-Molina [a], David Ruano-Ordás [a], Vitor Basto-Fernandes [b], José R. Méndez [a,*]

[a] Department of Computer Science, University of Vigo, ESEI - Escuela Superior de Ingeniería Informática, Edificio Politécnico, Campus Universitario As Lagoas s/n, 32004 Ourense, Spain
[b] Instituto Universitário de Lisboa (ISCTE-IUL), University Institute of Lisbon, ISTAR-IUL, Av. das Forças Armadas, 1649-026 Lisboa, Portugal

ARTICLE INFO

Keywords:
Ontology
Ontology fixing
Ontology quality measures
Ontology improvement methodology
Deming cycle

ABSTRACT

Ontology-learning methods were introduced in the knowledge engineering area to automatically build ontologies from natural language texts related to a domain. Despite the initial appeal of these methods, automatically generated ontologies may have errors, inconsistencies, and a poor design quality, all of which must be manually fixed, in order to maintain the validity and usefulness of automated output. In this work, we propose a methodology to assess ontologies quality (quantitatively and graphically) and to fix ontology inconsistencies minimizing design defects. The proposed methodology is based on the Deming cycle and is grounded on quality standards that proved effective in the software engineering domain and present high potential to be extended to knowledge engineering quality management. This paper demonstrates that software engineering quality assessment approaches and techniques can be successfully extended and applied to the ontology-fixing and quality improvement problem. The proposed methodology was validated in a testing ontology, by ontology design quality comparison between a manually created and automatically generated ontology.

1. Introduction and motivation

An ontology is originally defined as the philosophical study of being; it addresses questions related to which entities exist, and describes the categories and relations of these entities, as well as their hierarchisation and grouping criteria according to existing similarities and differences. Ontologies can be conceived following different paradigms and languages with different syntax, expressiveness, reasoning ability and models [1]. They are theories about objects, their properties, and relations that are defined in a domain of interest. The following definition given by Thomas Gruber [2] is generally accepted as the reference definition in the computer science community: "An ontology is a formal explicit specification of a shared conceptualization for a domain of interest". This definition is based on formal logic and allows for logic-based reasoning; it represents knowledge by the means of an explicit specification, and allows for a shared conceptualization of a domain of interest for a common and harmonized representation of knowledge (vocabulary, concepts, relations, etc.). In a global, fully interconnected world, ontologies are helpful in defining the basis for a common and shared understanding of data, information, and knowledge, both for people to machine or machine to machine communication and collaboration.

Web Ontology Language (OWL) [3] is a standard proposed by the World Wide Web Consortium (W3C) for defining ontology as a key building block for the Semantic Web. W3C defines a full semantic web protocol stack, including the eXtensible Markup

* Corresponding author.
E-mail address: moncho.mendez@uvigo.es (J.R. Méndez).

https://doi.org/10.1016/j.datak.2021.101889
Received 26 February 2020; Received in revised form 15 December 2020; Accepted 16 March 2021
Available online 26 March 2021

Language (XML) for the definition of a text-based document syntax/structure, Resource Description Framework (RDF) for the definition of concepts, classes, taxonomies, relations, and OWL to support the description logics inference ability. OWL ontologies represent an important step to ease the definition of a shared and common understanding of the structure of information in a domain, to enable the reuse of domain knowledge, and to systematically analyse domain knowledge with standard technologies on a global/internet/web scale. The transformation from a human-centred web content production (content produced to be consumed by humans) into a machine-readable web content production (content produced to be consumed/processed/understood by machines) has led to the development of a new set of methodologies, techniques, and tools. This new generation of technologies supports both the production of machine-readable content/knowledge from scratch in the form of OWL ontologies, and the transformation of existing content/knowledge (e.g. natural language text documents) into OWL ontologies.

The quantity of knowledge produced and made available on the Internet has grown exponentially in the last decades. Consequently, in order to fairly complete the analysis of existing knowledge in any specific domain of interest, the automatic and semantic processing of documents available on the web has become mandatory. As an example, in the domain of IoT (Internet of Things), there is an extensive catalogue comprising more than 400 ontologies [4].

With the passage of time, the need for ontologies to aid the semantic processing of documents has become more and more relevant. To quickly generate new ontologies at a low cost, several automatic ontology-learning methods [5–9] were introduced to extract knowledge from natural language text documents. Despite great advances in this research field, the use of these methods may result in the generation of inconsistencies and low-quality ontologies. This anomalous behaviour is directly connected with the intrinsic difficulties of different natural language processing challenges such as the disambiguation of word meanings (often called Word Sense Disambiguation, WSD) [10–13], handling informal text [12] or adequately dealing with new words from specific domains [13]. A direct consequence of this issue is that the costs of creating ontologies by using learning methods are not significantly reduced but are instead simply moved to a debugging/fixing stage.

Since fixing ontologies is usually a hard and manual task, the order in which errors are amended should be carefully selected to prioritize those ones resulting in a great improvement in the quality of the ontology, that implicitly solve other detected pitfalls. Moreover, automatic or semi-automatic fixing methods for the identified pitfalls (inspired in the quick fix actions implemented by most software-integrated development environments) would be a valuable support for this task. These issues suggest the design of a methodology (and some tools implementing it) to address the identification of ontology (semi)automatic defects and debugging/fixing operations. Although this kind of methodology could be successfully applied over any ontology, our proposal is especially suitable for addressing the correction of automatically generated ontologies, which usually contain a considerable amount of shortcomings. With this in mind, we found that methodologies and tools to handle inconsistencies could successfully complement automatic ontology learning methods, thus reducing the effort required to develop new ontologies from the collection of documents written in natural language.

To address the above-mentioned challenges raised by ontology learning methods, we defined a research framework and formulated the following research objective: (i) set an ontology learning workbench for automatic generated ontologies testing and research hypothesis validation; (ii) define a set of ontology quality metrics to support ontology quality assessment; (iii) define and implement a software-based mechanism for ontology redesign operations; (iii) formulate the ontology quality improvement problem as a multicriteria optimization problem; (iv) state and answer the research questions "Can an automatically generated ontology achieve similar quality of an ontology manually created by domain experts, based on the same source of knowledge (same natural language text documents)?", "How good are and what kind of assistance do ontology learning methods need to reach manually created ontologies quality?".

Taking the above-mentioned issues into account, this paper introduces a new methodology to address the identification and fixing of possible ontology pitfalls, inconsistencies, and/or errors, and to guide users to efficiently (in terms of time required) improve them. The proposed methodology is based on the well-known Deming cycle and incorporates elements extracted from OQuaRe, a SQuaRE (ISO 25000) based approach for evaluating the quality of ontologies. Additionally, the methodology includes graphical representations of ontology quality metrics to facilitate the ability to understand the quality level of an ontology at a glance.

The remainder of this paper is structured as follows: Section 2 provides an extensive review of previous studies on ontology quality assessment, ontology evaluation methodologies, and ontology automatic generation methods. Sections 3 and 4 introduce our proposal and the experimentation carried out to check its suitability. Finally, Section 5 compiles the main conclusions achieved upon carrying out this study, and future research possibilities related to this research line.

2. State of the art

Due to the interest of ontologies as a means of storing and exploiting the knowledge of different domains, research in several aspects of ontologies is extensive. In this section, we summarize the most important advances in the context of ontology learning (Section 2.1) and quality assessment (Section 2.2). Moreover, we find that some processes used to evaluate and help fix errors and defects in the context of software engineering could be successfully extrapolated for the domain of knowledge engineering. These concepts, processes, and technologies are compiled and explained in Section 2.3.

2.1. Ontology learning methods and tools

During the last years, many ontology generation tools and methods (also called ontology-learning methods) have been introduced with the main goal of reducing the effort involved in creating the ontologies [9,14]. This paper places special emphasis on (*i*) OwlExporter and LODeXporter, (*ii*) OntoPop, (*iii*) Text2Onto and (*iv*) XML module included in Protégé.

OwlExporter [3,15] can be included in a pipeline within the text-mining tool GATE (General Architecture for Text Engineering) [16,17]. OwlExporter allows exporting to document annotations in a web ontology language (OWL) model. Additionally, it is able to create and handle a domain-specific ontology to connect entities extracted from text to their lexical representation. In contrast with tools such as OwlExporter, other tools such as LODeXporter [18] present a complementary perspective of OwlExporter. LODeXporter is focused on the representation of domain data (e.g., biological entities, financial data), which is represented in domain-specific vocabularies, meant for creating Linked Open Data datasets and databases. Although these tools seem to be a great resource, they only provide an ontological population of the text from an existing ontology or an open vocabulary dataset. Therefore, these tools do not fully implement an ontology learning process.

OntoPop platform [19] implements a methodology. It aims to guide users in the integration of information extraction (IE) and knowledge representation (KR) tools in order to design domain-oriented applications for knowledge management. OntoPop performs successive attempts to integrate IE and KR tools, and to perform the annotation of documents and the population of ontologies on a corpus of representative text resources for the domain of the application. Hence, the OntoPop methodology defines a progressive and iterative framework with a defined termination condition that is reached when there is a common agreement among all users about the right integration of IE and KR tools. The OntoPop methodology comprises the following five stages: (*i*) study, (*ii*) structuring, (*iii*) mapping, (*iv*) validation and (v) delivery. During the study stage, the linguist, the knowledge domain expert, the ontology designer and engineer (sometimes called the knowledge engineer or simply the ontology designer) as well as the client evaluate the workload to adapt each tool to the domain. The second stage comprises the structuring of semantic labels resulting from IE in a conceptual tree and modelling of the domain ontology. During the third stage, each element defined in the domain ontology should be mapped with the semantic tags contained in the conceptual trees in order to create a set of Knowledge Acquisition Rules. During the validation stage, users should assess the quality of the annotations using the documents and knowledge base instances. Moreover, the integrator tests the implemented assignment while the client validates the general solution for the new application. Finally, during the last stage (delivery), the application is transferred to the customer and enters a state of maintenance.

Text2Onto [20] is an open source ontological learning framework developed to support the acquisition of ontologies from textual documents. It provides an extensible set of methods for the learning of atomic classes, subsumption and instantiation of classes, as well as object properties and axioms of disjunction. Text2Onto includes the following execution requirements: (*i*) a Java 1.6 virtual machine; (*ii*) GATE 4.0; and (*iii*) WordNet 2.0. The architecture of Text2Onto [20] is focused on the Probabilistic Ontology Model (POM) which stores the results of the different ontology learning algorithms. The algorithms are initialized by a controller, which also provides additional functionalities such as (*i*) triggering the linguistic pre-processing of the data, (*ii*) executing the ontology learning algorithms in the appropriate order, and (*iii*) applying the change requests of algorithms to the POM. The execution of each algorithm consists of three phases: (*i*) notification; (*ii*) computation; and (*iii*) result generation. During the first phase, the algorithm learns about recent changes in the corpus. During the second phase, these changes are mapped to changes with respect to the reference repository, which stores all kinds of knowledge about the relationship between the ontology and the data. Finally, in the last phase, requests for POM changes are generated from the updated content of the reference repository. However, Text2Onto proposes a semi-automatic process, which requires the expertise of ontology developers to validate the generated ontology, using some quality assessment methods (see next section).

Finally, Protégé is an open source tool to aid in the modelling of ontologies through the Protégé-Frames and Protégé-OWL editors. Ontologies can be exported to a variety of formats such as RDF, RDFS, OWL and XML Schema. Protégé was developed in Java and provides a plug-and-play environment that makes it a flexible base for the rapid development of prototypes and applications. In version 3.4, Protégé incorporates an XML module for importing files in XML format whose entries are used to create a set of classes and instances in a knowledge base [21,22].

The above-mentioned tools implement different ontology learning methods. Despite their usefulness, ontologies generated by using these methods are not free of errors and present design quality problems that need to be addressed a posteriori by design optimization approaches, eventually with human expertise assistance. The next section presents a revision of different methods to assess the quality of ontologies that can be used to detect the need of executing a fixing process or to provide relevant fixing information.

2.2. Ontology quality assessment metrics and methodologies

Keeping in mind the widespread use of ontologies to represent knowledge, the evaluation of their quality is currently a key aspect in their development and reuse. The results of a quality evaluation process allow the expert to recognize areas that might require additional work, cause design problems, or need to be fixed [23]. In this section, we present a review of different ontology quality evaluation approaches extracted from previous works.

There are different types of evaluations, including qualitative and quantitative methods [24]. A qualitative evaluation of an ontology can be complicated due to the limitations derived from the experience and the criteria of the ontological engineering experts. Moreover, knowing how and which evaluation parameters should be chosen is quite a difficult task. Therefore, in order to facilitate and automate the ontology evaluation, some authors [25] have proposed evaluation methods (Golden standard, Application based, Data-driven, Assessment by humans) defining different levels of quality such as (*i*) lexical, vocabulary, or data layer, (*ii*) hierarchy (also called taxonomy), (*iii*) other semantic relationships, (*iv*) context or level of application, (*v*) syntactic level, or (*vi*) structure, architecture and design.

The lexical, vocabulary, or data layer is focused on which concepts, instances or facts have been included in the ontology, and what vocabulary has been used to represent or identify these concepts. The hierarchy (or taxonomy level) refers to the hierarchical relationship between concepts. Other relationships may be also defined as the is-a relationship, which is used often and is particularly important in the evaluation process. In this level, measures such as precision and recall are generally included. The context (or application) level evaluates whether an ontology is part of a larger collection of ontologies, is referenced, or is an assessment point to take into account. Another form of context is the application in which the ontology will be used. This evaluation analyses how the results of the application are affected by the use of ontology. A syntactic level is especially targeted for ontologies that have been built manually. The ontology is usually described in a particular formal language and must match its syntactic requirements. Other syntactic considerations defined in [26] can also be considered. Finally, as with the previous level, the evaluation of structure, architecture and design is primarily used in manually constructed ontologies. At this level, the ontology should fit with certain predefined design principles or criteria.

The study of Bandeira *et al.*. [27] introduces FOCA, a methodology for the evaluation of ontologies. It is a three-step method, which comprises the identification of the type of ontology (i.e. Top Level, Domain, Task or Application ontologies), the application of a Goal/Question/Metric approach, and the assessment of the quality. Thus, FOCA implements a role-based calculation of the quality of the ontology according its type, comprises a questionnaire to accomplish the evaluation, and includes a statistical model that automatically computes the quality of the ontologies.

The OQuaRE [28] framework is a method to evaluate the quality of ontologies which adapt the SQuaRE standard (originally designed to assess the quality of software products) to ontologies. This framework defines all the elements required for ontology evaluation: evaluation support, evaluation process, and metrics. OQuaRE uses different metrics to assess the quality of the ontologies with regard to different dimensions, including reliability, operability, maintainability, compatibility, transferability, and functional adequacy. Most quality sub-characteristics suggested by SQuaRE (System and Software Quality Requirements and Evaluation) [29] were also adapted in OQuaRE. Additionally, OQuaRE includes the structural characteristic, which is important in evaluating ontologies.

FOval [30] introduces a new evaluation model to choose the ontology that best fits the user requirements. The model allows users to select indicators and assign weights for each one selected among a wide variety of available quality indexes. Moreover, FOval allows users to evaluate stored ontologies locally and/or find additional ontologies through the use of search engines.

OntoQA [31] is an approach that analyses ontology schemas and their populations (such as knowledge bases) and describes them through a well-defined set of metrics. These metrics can highlight the key features of an ontology scheme, as well as its population, and allow users to make an informed decision quickly. OntoQA evaluates the quality of an ontology on the different dimensions: schema, knowledge base (KB) and class metrics. This method can be used by ontology users before considering an ontology as a source of information, or by ontology developers to evaluate their work in building the ontology. Moreover, OntoQA [32] includes a suite of metrics to evaluate the content of ontologies through the analysis of their schemas and instances in different aspects such as the distribution of classes on the inheritance tree of the scheme, the distribution of class instances, and the connectivity between instances of different classes. One of the features that highlight OntoQA is its flexible technique for classifying ontologies based on their content and relevance to a set of keywords, as well as user preferences. OntoQA also evaluates ontologies using their instances (i.e. populated ontologies) and schemes.

Another work that contributes to the improvement of ontologies is OntoClean [33], a methodology focused on cleaning the taxonomy of ontologies based on the following perceptions [34]: *(i)* rigidity; *(ii)* identity; *(iii)* dependency; and *(iv)* unity. Rigidity refers to how essential a property is for all its instances. Three rigidity possibilities are defined as (+R) if and only if it is necessarily essential for all its instances, (-R) if and only if it is not essential for some of its instances, and (∼R) if and only if it is not essential for all its instances. Identity allows determining when a property has an identity criterion. There are two types of criteria (IC) (+I) and (+O). The first is valid only if all the instances of the property can be (re) identified by means of a suitable "sameness" relation. The latter is applied if and only if such criterion is not inherited by any inclusive property. On the other hand, dependency is when all the instances depend on a property P for their existence. Finally, unity refers to the relationship that unifies a set of parts to create a single individual. A property can carry the unit (+U) when a common relationship exists among all the instances. Conversely, a property can be anti-unit (∼U) when all its instances can possibly be non-wholes.

Appendix A contains an extensive list of quality evaluation metrics by studying a large set of ontology management tools. Our ontology quality assessment and improvement methodology takes advantage of all of the above-mentioned metrics (and evaluation areas). Moreover, our proposal also brings innovative ideas from software quality evaluation area to the area of knowledge engineering. The next section details software engineering techniques that have contributed to interesting solutions developed in the context of this work.

2.3. Evolution from software engineering quality metrics to ontology quality indicators

This subsection underscores the software engineering issues and solving methods that have been successfully adapted to the domain of knowledge engineering. A notable example is the OQuaRE framework (see the previous subsection) whose metrics have been inspired, adapted and derived from similar metrics included in the SQuaRE software engineering standard. Keeping in mind the close connections between software engineering and knowledge engineering, this subsection compiles several studies that take advantage of the software engineering process to solve knowledge engineering challenges.

Commonly used in the software development area, the term 'pitfall' refers to the common bad practices that usually appear during the process of designing and building applications. These anomalies can be identified by evaluating simple or composed conditions related to ranges or thresholds of specific metrics (e.g. the number of lines inside a class). Pitfalls are also known in the literature as Code Smells [35] (or simply smells) and represent symptoms (potential defects) that reveal potential software design problems. For instance, a Blob Class (sometimes called God Class) is a large class having a high variety of responsibilities that may not constitute an error but shows an important design issue. Other well-known symptoms are Duplicated Code, Long Method, and Lazy Class, among others. A source code affected by code smells (or pitfalls) should be fixed through refactoring actions, in order to improve code design, readability, comprehension, maintainability, and other relevant issues.

Researchers quickly realized that bringing the concept of smells/pitfalls to the domain of knowledge engineering could help in the process of improving the quality of ontologies. The work of Póveda-Villalón et al. [36] introduces and classifies a catalogue of 24 pitfalls that usually appear during the process of building ontologies. The classification of pitfalls is based on several works prepared by different authors who have identified common mistakes during the process of modelling ontologies. One of the main objectives of this work was to group pitfalls using two different criteria: (*i*) dimensions of structure, function and usability; and (*ii*) aspects of consistency, completeness, and conciseness.

OOPS! [37] is a web application that detects bad practices causing errors in the modelling of ontologies. This tool provides mechanisms for the automatic detection of potential errors, called pitfalls, in order to help developers during the validation process. However, some pitfalls are detected semi-automatically, such as "Creating synonyms as classes" and "Creating unconnected ontology elements", among others [37]. Each pitfall provides the following information: title, description, elements affected and importance level. For the evaluation of the results, two types of classifications are presented: dimension and criteria. When classifying by dimension, pitfalls are divided into certain categories to evaluate the ontologies in relation to their structure, functionality, and usability profile. On the other hand, the classification by criteria establishes the following criteria: consistency, completeness, and conciseness.

Additionally, software refactoring has emerged as a set of strategies and tools to improve the design of existing (and sometimes fully functional) software applications [38,39]. Specifically, refactoring [38] is the process of improving code design and minimizing the possible appearance of future bugs, without interfering with the external behaviour of the software. It suggests guidelines for solving code design problems through refactoring. However, it is up to the developer's criteria or intuition to know the course of action that must be taken; for example, how many instances to change or how many lines of code are sufficient. Although this concept has not yet been brought to Knowledge Engineering, we believe that these ideas could be successfully adapted to this domain (ontologies) to solve ontology pitfalls/smells.

The above-mentioned concept (refactoring) is intimately related to the code quick-fix utilities. In fact, whilst refactoring emerged to facilitate the improvement of software design in existing software systems, code quick-fix and recommendation tools (e.g. jDeodorant) included in popular IDEs (Integrated Development Environments) aid software developers in automatically solving compiler errors or software design and implementation defects [40–44]. Given the similarity between code error and ontology inconsistency, quick-fixes could be successfully applied to knowledge engineering to automatically fix inconsistencies detected in ontologies by running recommended automated actions.

The next section presents our methodology, which brings together the technologies introduced in this subsection to aid ontology designers in the challenging task of fixing ontologies.

3. Methodology

This section introduces our methodology to assess the process of correcting ontologies. It is based on reusing the well-known Deming cycle [45], also known as PDCA (Plan - Do - Check - Act), which guides most quality processes. The PDCA cycle makes it possible to easily address the continuous process of improvement in a 4-stage repeating system. For this reason, it has been adopted as working framework in a wide range of ISO/IEC standards and other quality assessment/improvement proposals [46–49]. We find that this well-known cycle can successfully guide the iterative improvement of a target ontology. Fig. 1 shows an overview of our proposal with summarized information about the elements comprised in each stage.

As shown in Fig. 1, our methodology iterates through four different stages. In detail, stage one (see Plan in Fig. 1) handles the application of different metrics to measure the quality of the ontology. The second stage (defined as Do in Fig. 1) is responsible for (*i*) detecting defects in the ontology and (*ii*) ranking them according to their relevance. Once the most relevant issue has been selected, the next stage (see Check in Fig. 1) involves an assessment (automatically or with an expert) of the feasibility of solving the issue. Finally, the last stage (called Act in Fig. 1) executes the most adequate ontology fixer mechanism according to the information collected from the previous stage.

The following subsections provide a detailed description of each of the stages involved in our methodology lifecycle.

3.1. Plan

During this stage, the ontology should be evaluated using quality metrics to check its consistency and design harmony. The results of the evaluation will allow a decision on whether a new cycle of the methodology is executed or not. When making this decision, an ontology designer should bear in mind the available time to complete a new cycle and obtain quality results.

In order to evaluate the quality at a single glance (graphically) we find it adequate to represent the metrics RROnto, INROnto, ANOnto, CROnto, NOMOnto, RFCOnto, CBOOnto, LCOMOnto and RCOnto (see Section 2.2) in a radar chart (which can be created using a variety of libraries[1]). The area of the figure described by the representation of these measurements could be easily interpreted as the global quality of the ontology.

[1] See https://www.amcharts.com/demos/radar-chart/

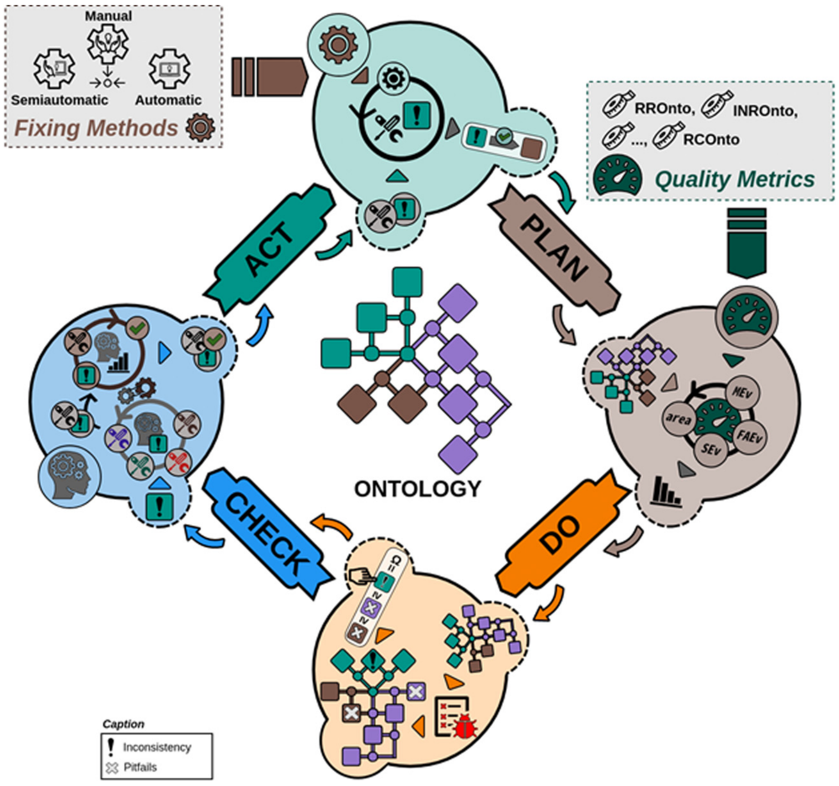


Fig. 1. Methodology lifecycle.

Moreover, in order to evaluate different aspects of the ontology in a separate form, it is possible to compute the ontology scores implemented in the OQuare framework [28]: (*i*) SEv represents its structural evaluation; (*ii*) FAEv addresses the evaluation of functional adequacy; and finally (*iii*) MEv evaluates its maintainability. These scores have been designed as an average of the scores of their associated sub-characteristics identified in Section 2.2. SEv, FAEv and MEv can be easily computed using Eq. (1).

$$\begin{aligned} &SEv = AVG\,(RROnto, ANOnto, LCOMOnto)\\ &FAEv = AVG\,(ANOnto, RROnto, INROnto, LCOMOnto, CROnto, NOMOnto)\\ &MEv = AVG(LCOMOnto, NOMOnto, CBOnto) \end{aligned} \tag{1}$$

Moreover, according to the OQuare framework, adding *SEv, FAEv* and *MEv* scores allows us to compute another overall score of the quality of the ontology, thus aiding in the decision of executing a new cycle of the methodology.

3.2. Do

In this stage, we should identify, select, and analyse a concrete target problem to solve in the ontology. The existence of ontology errors and/or pitfalls is the main reason of achieving poor results in its quality evaluation. As an example, the existence of an incompleteness error (which is included in structural and functional categories) implies not representing all the knowledge that could be included in the ontology (e.g. annotations) which causes an impact on the value obtained for the ANOnto structural/functional metric. Similarly, the existence of other errors and pitfalls will have an impact on this or other quality measures used. Despite the quality metrics allows to understand the worth of an ontology at a glance, they do not allow to identify concrete issues that should be addressed to improve the target ontology. Therefore, one of the most important things in the methodology is to automatically identify and rank problems to facilitate the selection of a relevant issue whose resolution could result in a high positive impact in the quality of the ontology. In this sense, inconsistencies are the most relevant issues that can be found in ontologies because they would prevent using and reasoning on the ontology knowledge. Therefore, inconsistencies are the highest priority issues to be addressed and solved.

Next, pitfalls and smells should be addressed in order to improve the quality of the ontology. In our work, we take advantage of the pitfalls identified in previous literature [35]. These pitfalls are identified by using the OOPS! framework [37] and the error rating (*importanceLevel*, defined as Ω in Fig. 1) provided by this software. Three levels are defined: ((i) *critical* is the most important level to correct because it affects the consistency of the ontology; (*ii) important* is not critical for the operation of the ontology, although it is important to correct this type of pitfall because it affects the quality of the ontology; and (*iii) minors,* which does not indicate a problem, although correcting it improves the appearance of the ontology.

The list of issues (inconsistencies and pitfalls/smells) are defined in the same order as previously described in the output of this stage. Despite the fact that a complete list is initially identified, only a single issue is solved in each cycle of the methodology.

3.3. *Check*

This step is responsible for solving the error found in the previous stage (ranked by importance). Sometimes (when supported by available tools) one or several quick-fix(es) could be applied to address the selected issue. When quick fixes are available, quality metrics could provide a reasonable way of deciding the best quick-fix method to solve the issue. In this kind of situation, the area of the original radar chart (see stage plan in Section 3.1) could be compared with the area of other ontologies achieved as a result of applying each quick fix. Ordering these quick fixes by the improvement achieved (the difference between the quality of the ontology achieved by applying the quick fix and the original ontology) could provide information for the ontology designer. However, ontology designers, guided by their experience, could decide to apply more than one quick fix, to use a different method, or even to discard the issue (if it is not an issue). When quick fixes are not available, ontology designers should make the most of their experience and decide on a method to fix the selected issue.

In the case of smells or pitfalls, the selected issue is sometimes discarded/ignored because it is not really an error but a design option conscientiously made by the knowledge engineer.

3.4. *Act*

We should implement the ontology fix. There are three different types of fixes: (*i*) automatic, (*ii*) semiautomatic and (*iii*) manual. If the fix is automatic, it can be easily applied. However, most times the issues should be solved manually. Over time, new tools will emerge and provide automatic implementations (quick fixes) to solve many issues, thus facilitating this task and, ultimately, the whole process.

The application of this methodology will make it possible to take advantage of ontology learning methods to produce ontologies with better quality. The next section presents the experimentation carried out in this work and shows the utility of the defined methodology.

4. Case study

This section provides a detailed description of the experiments performed to demonstrate the utility of this work. Section 4.1 compiles a list of ontologies freely distributed online that could be used to execute a case study of the use of this methodology. Section 4.2 presents a detailed application of the introduced methodology to fix different kinds of defects. Finally, Section 4.3 shows a list of lessons learned from the process of designing the ontology.

4.1. *Publicly available ontologies*

Currently, there are different sites where it is possible to find ontologies developed in different domains that are made available to the community. These sites give the experts the possibility to share their own ontologies, extend, or even comment on previously developed ontologies in order to share feedback among the ontology developers. Below is a brief description of the sites that allow open ontologies to be downloaded.

BioPortal [50] is a site that provides an open repository consisting of a broad set of ontologies in the biomedical domain. It also provides access to tools to work with the ontologies (e.g. tools to receive recommendations about which ontologies are the most relevant for a corpus, to annotate texts with terms from ontologies, to search for biomedical resources for a term, etc.). Agroportal [51] is a site that allows finding and sharing ontologies in the area of agriculture. It also allows performing reviews and comments on ontologies and their components while navigating. Ontohub [52] is a site which has approximately 128 repositories available from different domains. Featured repositories are Basic Formal Ontology (BFO), which contains 72 ontologies, Common Logic Repository COLORE with 2,653 ontologies, and FOIS Ontology Competition with 50 ontologies. OBO Foundry [53] has developed a set of ontologies in the Biological and Biomedical domain that are available on their website. Finally, Swoogle [54] is an ontology search engine that allows the user to enter a word or keywords to find the related ontologies available on the web.

Despite the relevance of the above-mentioned sites where we can find many ontologies, we found an interesting work authored by Li et al. [55] that introduces a full-featured hand-made ontology to represent the knowledge of Preference-Based MultiObjective Evolutionary Algorithms (PMOEA) present in 62 original scientific papers that have been represented manually in the ontology. The main interest of this ontology is the availability of the knowledge domain experts and ontology designers to share the rationale and experience behind their design decisions for this specific case. Therefore, we could execute an automatic ontology learning process from these works, fix the resulting ontology, and use the hand-made ontology for comparison purposes. We found that the results of fixing this automatically generated ontology would lead to a valuable example for the methodology introduced in this work.

Table 1
Definitions algorithms Text2Onto.

Algorithms	Component	Description
TFDIFConceptExtraction	Concept	It calculates term frequency inverse document frequency which is the product of Term Frequency (TF) and Inverse Document Frequency (IDF).
TFIDFInstanceExtraction	Instance	Is similar to TFIDFConceptExtraction. It computes TFIDF of each instance and then normalizes them in the same fashion.
WordNetClassifcationExtraction	SubClassOf	It extracts subclass-of relations among the extracted concepts identifying the hypernym structure of the concepts in WordNet.
SubcatRelationExtraction	Relation	The algorithm SubcatRelationExtraction identifies the following syntactical frames: *Transitive, Intransitive + PP -complement, Transitive + PP-complement.* For each verb phrase, it finds its subject, object, and associated preposition.
PatternDisjointClassesExtraction	Disjointness	A heuristic approach based on *lexico-syntactic* patterns is implemented to learn disjointness. The algorithm learns disjointness from the patterns like: $NounPhrase_1$, *NounPhrase*$_2$, …., *(and/or) NounPhrase*$_n$

4.2. The target ontology

As mentioned above, we designed an automatic learning process to build an ontology from 62 specific scientific articles in the domain of "MultiObjective Optimization". Additionally, we were able to take advantage of the manually generated ontology to assess the results achieved. In this subsection, we describe the process used to automatically learn the new ontology.

For the extraction of information from the articles, tests were done with the different tools that are used for extracting information from scientific articles in PDF format. In particular, the PDFX [56], CERMINE [57], Sapient [58], Parscit [59], and LA-PDFText [60] platforms were analysed in detail. For each of these tools, we checked its online availability and proper operation (i.e. the correct extraction of documents in different versions of Portable Document Format). Based on these results, we found CERMINE to be the most reliable for our needs. The file generated by this tool is an XML with a logical structure that classifies and perfectly defines the different sections of a scientific article (authors, introduction, abstract, conclusion, etc.) which helps to identify and manipulate the most important parts of the document. The generated XML file was later processed by a Java application (developed by the authors of this paper using JDOM API[2]) to transform the relevant sections of the XML document into a text file to be used as an input corpus for the automatic ontology generation tool. Additionally, we analysed several automatic ontology learning tools including OwlExporter, OntoPop, Text2Onto and Protégé. All of them are available online and have been described in Section 2. After the comparison of the tools, Text2Onto was selected since it was one of the tools that can be easily adapted and automatically generates the ontology through a corpus provided. In addition, Text2Onto includes various (configurable) algorithms to improve the ontological output.

According to the selected configuration, the information of the 62 papers was extracted with the Cermine tool. Then, we processed the obtained XML documents to extract and store the following parts in a text file: (*i*) abstract, (*ii*) the contents of the introduction section, (*iii*) the author list and (*iv*) the conclusions section.

The parameters optimization of Text2Onto tool [61] was done using a subset of 10 of the 62 documents from the original dataset. This task was experimentally made by analysing various combinations of the available algorithms for guessing concepts, instances, relations "subclass of", other kinds of relations and disjointness. Table 1 summarizes the combination of algorithms that we finally selected.

The descriptions of the selected configurations are available in the original work of Mittal [61]. Additionally, we manually selected the terms from texts that would be represented in the resulting ontology (as concepts, instances, relationships, etc.) in POM view. This task was not automated because we found several linking-words in the set of automatically selected terms. Table 2 shows the basic metrics computed by the Protégé tool of the automatically generated ontology vs manually generated ontology (number of classes, subclasses, and instances, among others).

A glance at the basic metrics computed for each ontology shows a big difference in the number of axioms, classes, instances, subclasses and disjoint classes between the automatic and manual ontologies. We have many more elements in the automatic ontology because we cannot automatically filter some irrelevant concepts such as "*title*" or "*figure*". Additionally, ontology graph representations for both ontologies are supplied as Figure B1 in Appendix B.

4.2.1. Results achieved for plan phase

To assess the quality of the selected ontologies we combined structural, functional, adequacy and maintainability metrics in a radar chart, as seen in Section 3.1. Fig. 2 shows the score obtained by each of the ontologies in each metric. Additionally, we computed *SEv*, *FAEv* and *MEv* metrics (introduced by OQuare) in accordance with Section 3.1 and compiled their results in Table 3.

[2] Available at http://www.jdom.org

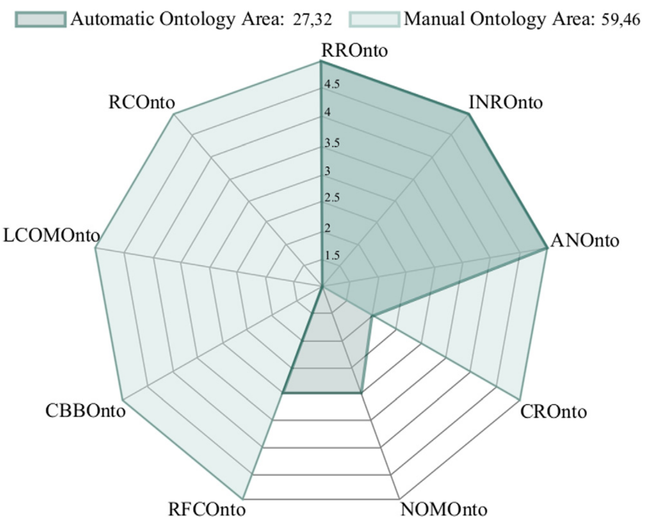


Fig. 2. Results of the ontologies quality evaluation.

Table 2
Base metrics of the automatic and manual ontology.

Metrics	Automatic ontology	Manual ontology
Axioms	14003	3894
Logical axioms count	3703	2441
Declaration axioms count	2146	634
Classes count	2042	92
Object properties count	102	16
Individuals count	575	16
DL expressivity	ALC	510
Class axioms		
SubClass Of relations	2225	81
Equivalent Classes	0	2
Disjoint Classes	699	9

Table 3
SEv, FAEv and MEv metrics.

Metric	Sub-characteristic	Weighted average automatic ontology	Weighted average manual ontology
Structural	*AVG(RROnto, ANOnto, LCOmonto)*	3.66	5
Functional adequacy	*AVG(ANOnto, RROnto, INROnto, LCOMOnto, CROnto,NOMOnto)*	3.5	5.16
Maintainability	*AVG(LCOMOnto, NOMOnto, CBOnto)*	1.66	3.66
GLOBAL VALUE (SEv + FAEv):		3.58	5.08

To obtain an overall score of the ontologies and compare them with each other, the OQuare model was taken as a reference, which adds the metrics of structure and functional adequacy, as shown in Table 3. As we can see from the results shown in Fig. 2 and Table 3, the quality differences between the two ontologies can be easily noted. As a result, we can observe that the manual ontology obtained higher scores with respect to the automatic ontology. This indicates that the automatic ontology requires quality improvement to be used, understood, and learned. However, in some metrics, such as relationship richness and coupling between objects, equal values were obtained. This analysis can help ontology designers to detect the parts in which the ontology can be improved.

Table 4
Pitfalls found using the automatic ontology.

Dimension	Description	Importance
Circularity	Including cycles in a class hierarchy	Critical
Incompleteness	Using different naming conventions in the ontology	Minor
Semantic	Creating unconnected ontology elements	Minor
Incompleteness	Missing annotations	Minor
Incompleteness	Inverse relationships not explicitly declared	Minor

4.2.2. *Results of the Do phase*

The methodology includes a step to identify ontology issues (inconsistencies and/or pitfalls) and their causes. This will allow the user to correct the ontology and turn it into a better one. The first step should be focused on fixing inconsistencies to allow an ontology to be processed by a reasoner. None of the evaluated ontologies (manual and learned) contains inconsistencies. Therefore, with the independence of their quality, these ontologies can be used by a reasoner.

In this phase, we should also identify pitfalls using the work of Poveda Villalón et al. [36] as a starting point and sort them according to the level of importance in which they must be attended. This procedure will help the user to identify the classes and axioms that can be eliminated to improve the quality and design of the ontology. Table 4 shows the results of the pitfall/error detection stage.

As can be deduced from Table 4, the most critical ontology issue (that with the highest priority to be addressed) is the circularity error described in the first row of the table. Circularity occurs when there is a cycle between two (or more) classes; for example, some class A has a subclass B and at the same time, B is a superclass of A. A particular example of circularity was found when analysing the classes *relation* and *action*. The IS_A relations for these classes are *relation* IS_A *action*, *action* IS_A *relation* conforming a cycle.

4.2.3. *Results of the check phase*

To accomplish this stage, we will take advantage of different quick fix methods that would automatically fix errors and smells. The stage involves computing the effects of applying each quick fix available for addressing the selected ontology issue in terms of quality improvement, new errors generated, and edition operations involved to execute the quick fix.

To cope with circulatory issues, we introduced the RM_INVOLVED_ELEMENTS quick fix, consisting of removing some (or all) the axioms related to the ontology elements (classes, object properties or data properties) that are causing the trouble. Furthermore, we introduced RM_SIMILAR_ELEMENTS quick-fix that searches and removes similar elements caused by typos that are causing a pitfall in the ontology. For this purpose, we take advantage of the Levenshtein algorithm [62] to find the lexical distance between two words. This quick-fix removes elements having a distance lower than or equal to 2. For example, when comparing the elements "action" and "section", which could cause circularity in the ontology, the distance calculated between them is 2; that is, there is a similarity between the terms such that when applying the RM_SIMILAR_ELEMENTS quick fix, one of the elements is removed from the ontology in conjunction with the axioms related to it.

Fig. 3 shows detailed information about the quality results achieved if RM_INVOLVED_ELEMENTS and RM_SIMILAR_ELEMENTS are applied together with the list of operations that would involve their application.

The main goal of this stage is to select the most appropriate strategy to deal with the error/pitfall that is being fixed through the information computed. As shown in Fig. 4, when selecting RM_TYPOS, the circularity issue would be partially solved, but it adds a new important error (incompleteness) to the ontology. On the other hand, RM_INVOLVED_ELEMENTS seems to achieve a better improvement of quality because it completely removes the circularity issue, although a new critical error is also added. Fig. 4 shows in detail the classes that form a loop, especially relation-action and type-case.

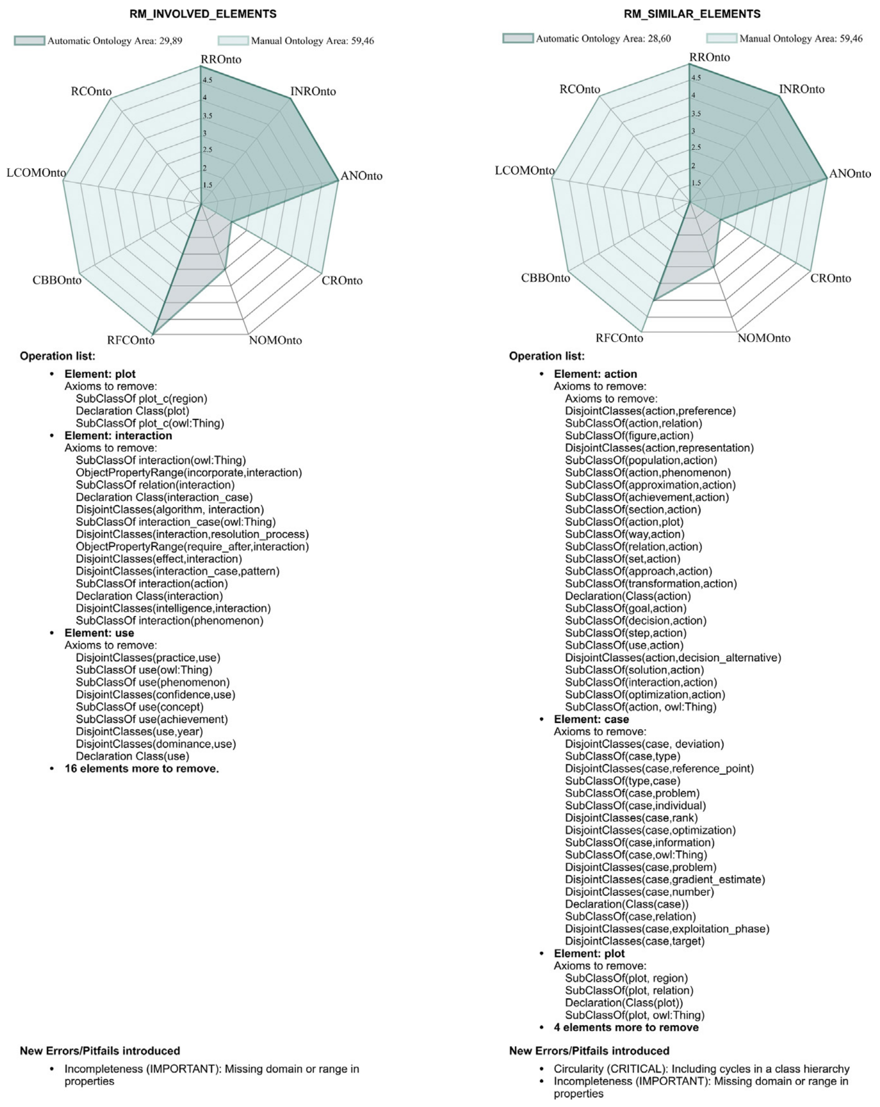
Additionally, as noted at the bottom of Fig. 3, the new pitfall (incompleteness) added when applying any quick fix has an important level of criticality that indicates "Missing domain or range in properties". In other words, it is necessary to relate some properties to a domain or range of the ontology. On the other hand, newer errors introduced by RM_SIMILAR_ELEMENTS are more critical than those found using the RM_INVOLVED_ELEMENTS quick fix.

After comparing each of the quick-fixes we decided that the RM_INVOLVED_ELEMENTS is the best solution for the case study because we obtain better results in relation to the area of the radar chart, which reflects the quality of the ontology and, moreover, it completely removes the circularity pitfall.

4.2.4. *Results of the act phase*

In this phase we apply the quick fix (or the action) selected in the previous phase. Once the quick fix is applied, a new version of the ontology is created with the new changes, and a copy of the original ontology is saved to allow restoring it to its original state.

Additionally, to solve a pitfall, a manual edit is also possible. According to the detected troubles, the expert applies a subjective-criteria, chooses which actions to perform in order to improve the ontology based on his or her criteria, and executes them through



Operation list:

- **Element: plot**
  Axioms to remove:
  SubClassOf plot_c(region)
  Declaration Class(plot)
  SubClassOf plot_c(owl:Thing)
- **Element: interaction**
  Axioms to remove:
  SubClassOf interaction(owl:Thing)
  ObjectPropertyRange(incorporate,interaction)
  SubClassOf relation(interaction)
  Declaration Class(interaction_case)
  DisjointClasses(algorithm, interaction)
  SubClassOf interaction_case(owl:Thing)
  DisjointClasses(interaction,resolution_process)
  ObjectPropertyRange(require_after,interaction)
  DisjointClasses(effect,interaction)
  DisjointClasses(interaction_case,pattern)
  SubClassOf interaction(action)
  Declaration Class(interaction)
  DisjointClasses(intelligence,interaction)
  SubClassOf interaction(phenomenon)
- **Element: use**
  Axioms to remove:
  DisjointClasses(practice,use)
  SubClassOf use(owl:Thing)
  SubClassOf use(phenomenon)
  DisjointClasses(confidence,use)
  SubClassOf use(concept)
  SubClassOf use(achievement)
  DisjointClasses(use,year)
  DisjointClasses(dominance,use)
  Declaration Class(use)
- **16 elements more to remove.**

New Errors/Pitfalls introduced

- Incompleteness (IMPORTANT): Missing domain or range in properties

Operation list:

- **Element: action**
  Axioms to remove:
  Axioms to remove:
  DisjointClasses(action,preference)
  SubClassOf(action,relation)
  SubClassOf(figure,action)
  DisjointClasses(action,representation)
  SubClassOf(population,action)
  SubClassOf(action,phenomenon)
  SubClassOf(approximation,action)
  SubClassOf(achievement,action)
  SubClassOf(section,action)
  SubClassOf(action,plot)
  SubClassOf(way,action)
  SubClassOf(relation,action)
  SubClassOf(set,action)
  SubClassOf(approach,action)
  SubClassOf(transformation,action)
  Declaration(Class(action)
  SubClassOf(goal,action)
  SubClassOf(decision,action)
  SubClassOf(step,action)
  SubClassOf(use,action)
  DisjointClasses(action,decision_alternative)
  SubClassOf(solution,action)
  SubClassOf(interaction,action)
  SubClassOf(optimization,action)
  SubClassOf(action, owl:Thing)
- **Element: case**
  Axioms to remove:
  DisjointClasses(case, deviation)
  SubClassOf(case,type)
  DisjointClasses(case,reference_point)
  SubClassOf(type,case)
  SubClassOf(case,problem)
  SubClassOf(case,individual)
  DisjointClasses(case,rank)
  DisjointClasses(case,optimization)
  SubClassOf(case,information)
  SubClassOf(case,owl:Thing)
  DisjointClasses(case,problem)
  DisjointClasses(case,gradient_estimate)
  DisjointClasses(case,number)
  Declaration(Class(case))
  SubClassOf(case,relation)
  DisjointClasses(case,exploitation_phase)
  DisjointClasses(case,target)
- **Element: plot**
  Axioms to remove:
  SubClassOf(plot, region)
  SubClassOf(plot, relation)
  Declaration(Class(plot))
  SubClassOf(plot, owl:Thing)
- **4 elements more to remove**

New Errors/Pitfalls introduced

- Circularity (CRITICAL): Including cycles in a class hierarchy
- Incompleteness (IMPORTANT): Missing domain or range in properties

Fig. 3. Results of the check phase.

an external tool such as Protégé. Once the modifications have been made, the expert can establish the next state of the ontology and compute quality metrics to verify whether an improvement has been achieved (the area of the radar chart is bigger).

In the example proposed, RM_INVOLVED_ELEMENTS fix was applied to all the elements that were causing circularity. Through this automatic quick-fix, we achieved an area of 29.89 (previous area 27.32) which confirms a quality improvement (specially in RFCOnto metric). It is important to mention that when applying a quick fix, there is a possibility of adding a new pitfall to the ontology. In this case a pitfall with an important level of criticality and a total of 19 elements that affect incompleteness were added.

4.3. *Advantages of our proposal*

After having explained the goals of each stage included in the methodology, this subsection highlights some of the advantages of our proposal. In particular, the use of quick-fixes, the possibility of returning the ontology to an earlier state, the ease of understanding the techniques (because they are completely inspired in software engineering) and the possibility of automating its application (with undo support).

Quick fixes can be applied several times in order to eliminate many errors and improve the quality of the ontology. In the case study, the quick fix was applied a second time in order to correct the incompleteness pitfall. After recalculating the quality metrics there was an improvement in the area of the automatically generated ontology, with a new value of 32.14 (Fig. 5) compared to the previous area of 29.89. By applying this quick-fix we eliminated those pitfalls that had a critical and important impact on

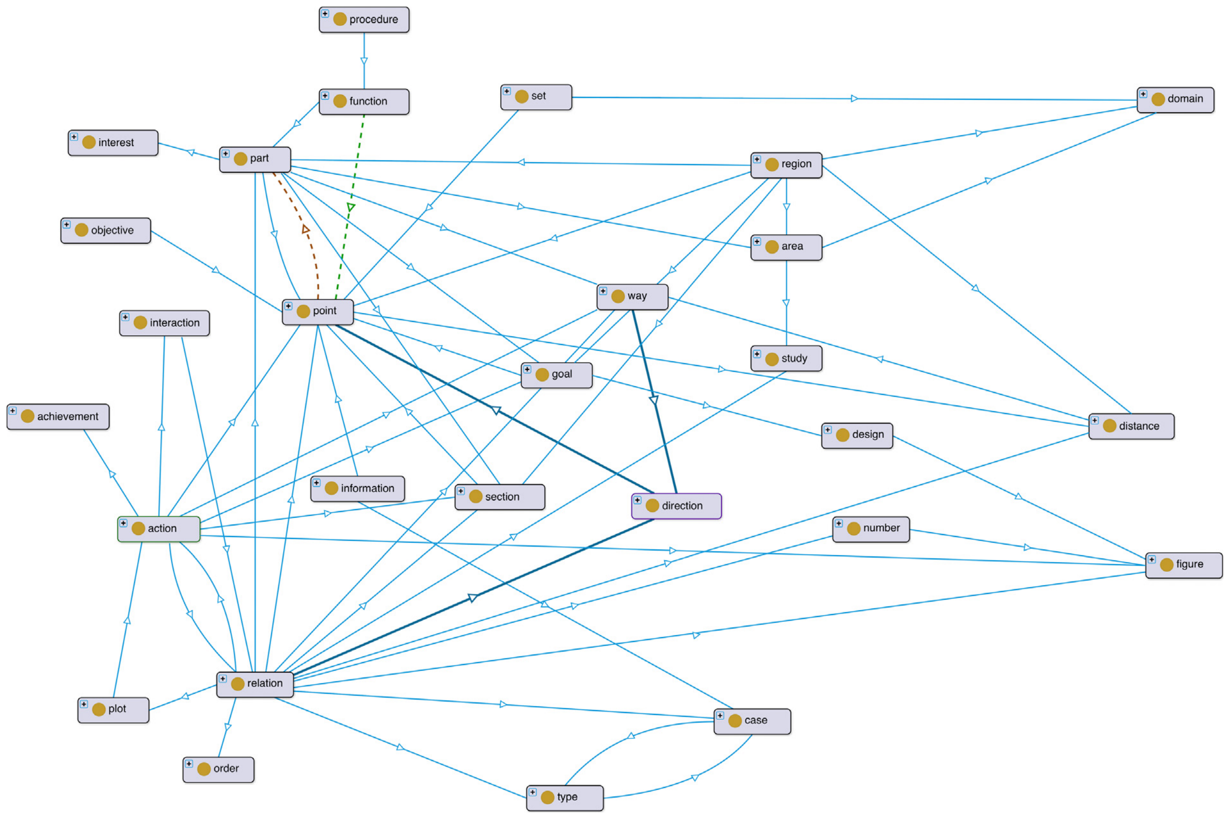


Fig. 4. Classes causing circularity.

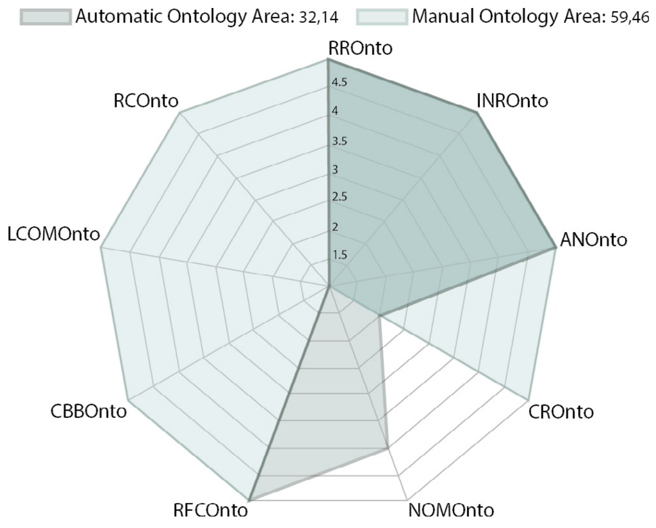


Fig. 5. Results of applying *RM*_INVOLVED_*ELEMENTS* quick fix again.

the ontology, and we improved the quality of the ontology by 19% compared to the first version of the automatically generated ontology. This has allowed us to verify that the removal of the elements that cause a problem in the ontology improves its quality.

Another advantage to mention is the possibility of returning the ontology to an earlier version after a quick fix has been applied. In addition, the expert can view the history of the changes made and which elements (axioms) were removed for improvement of the ontology. We also want to emphasize that our proposal is scalable; that is, there is the possibility of adding new quick fixes as well as new metrics that help to better define the quality of the ontology.

The automatic and manual ontologies differ significantly. This suggests that generating high quality ontologies by only using ontology learning tools is a difficult task. Therefore, the definition of methodologies (such as that defined in this work) is absolutely necessary. Due to the parallelism between software and knowledge engineering domains, we find it adequate to use all mechanisms from the former domain to successfully address the troubleshooting process of the latter. The focus of this paper was to show the suitability of quick fix methods to improve the quality of ontologies. The advantage of utilizing software engineering techniques is that they are commonly known and easier to understand.

We would like to highlight that the application of the proposed methodology could be easily automated by developing a support software application. A web application developed by the authors for ontology quality assessment and improvement is publicly available on https://github.com/gabyluna/OntologyFixer. The software automatically execute each stage and allow reverting the changes to previous ontology states.

To summarize the contributions of our work, we can state that our methodology and corresponding mechanisms provide an efficient user involvement in (semi)automatic ontology quality improvement, being an essential complement research to the ontology learning methods and tools.

Ontology learning methods need to be validated by the users to ensure their quality, which raises a challenge on minimizing user involvement and maximizing user involvement reflex on final ontology quality. As shown above, our methodology and mechanisms address this challenge in the following dimensions: (*i*) ontology inconsistency detection and fixing; (*ii*) ontology quality assessment by the means of a variety of widely accepted metrics in the knowledge engineering area; (*iii*) define a mechanism for implementing ontology redesign operations; (*iv*) allow for user comprehension on (semi)automatic improvement actions reflexes on ontology quality metrics; (*v*) consider user (knowledge engineer) preferences in the ontology quality improvement methodology and mechanisms and (*vi*) focus user effort on the most effective ontology fixes and redesign change possibilities.

5. Conclusions and future work

This paper has presented a methodology to address the process of fixing ontologies (especially those created through the use of ontology learning methods) that make extensive use of software engineering technologies. The improvement of quality is guided by well-known quality metrics (through the area contained in a radar chart representing them) and the application of quick fix elements to automatize simple changes in the target ontology and improve its quality.

Our methodology is inspired in the use of the well-known Deming cycle (PDCA) used to guide quality standards. During the Plan stage, the target ontology is evaluated to detect and sort errors or pitfalls that should be addressed. During the Do stage, the user selects which of the available troubles should be addressed first. The Check process comprises the evaluation of different quick fixes to address the selected trouble, and the election of the most appropriate one. Finally, the Act includes the application of the selected quick fix or the manual edition of the ontology to perform the required changes.

The current methodology comprises the use of radar charts combining different quality metrics and the area it describes to assess the global quality level of the methodology. This representation facilitates the observation of ontology quality levels at a glance, and helps users to visually determine the main deficiencies of the ontology. Additionally, the methodology comprises the determination of the errors and pitfalls (smells) that need to be addressed and provides a mechanism to sort them using their importance as criterion. This functionality helps users to select the most appropriate error to fix in each methodology cycle.

The experimental results show that both the methodology and the defects detection and fixing proposed strategies can significantly reduce the user's workload, save time, and increase the resulting ontologies quality. Our approach reveals an innovative contribution to the state-of-the-art on semi-automatic ontology quality improvement.

For future work, we will develop new quick fix strategies and new forms of ontology optimization that help improve the quality of ontologies. Moreover, we are currently developing an application to fully automate the use of the methodology described in this paper. This application will provide support for the decision-making process of users regarding the actions that can be applied to improve the quality of ontologies that were created automatically.

CRediT authorship contribution statement

Gabriela R. Roldán-Molina: Executed the methodology process (quality measurements and quick-fix execution), Developed some quick-fix methods. **David Ruano-Ordás:** Contributed some text for the manuscript, Made figures, Give advice about text processing issues. **Vitor Basto-Fernandes:** Conceived the idea, Wrote some parts of the manuscript, Supervised the work done. **José R. Méndez:** Conceived the idea, Wrote some parts of the manuscript, Supervised the work done.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

D. Ruano-Ordás was supported by a post-doctoral fellowship from Xunta de Galicia (ED481B 2017/018). Additionally, this work was partially funded by Consellería de Educación, Universidades e Formación Profesional (Xunta de Galicia), Spain under the scope of the strategic funding of ED431C2018/55-GRC Competitive Reference Group.

This work was partially supported by the project Semantic Knowledge Integration for Content- Based Spam Filtering (TIN2017-84658-C2-1-R) from the Spanish Ministry of Economy, Industry and Competitiveness (SMEIC), State Research Agency (SRA) and the European Regional Development Fund (ERDF).

SING group thanks CITI (Centro de Investigación, Transferencia e Innovación) from University of Vigo for hosting its IT infrastructure.

Appendix A. Supplementary Materials

Supplementary materials related to this article comprises two appendices and can be found online at https://doi.org/10.1016/j.datak.2021.101889. Appendix A contains a study of metrics included in different frameworks used to assess ontology quality. Moreover, appendix B provides graphical representations of ontologies used in this study.

References

[1] G. Barchini, M. Álvarez, S. Herrera, Sistemas de información: nuevos escenarios basados en ontologías, JISTEM - J. Inf. Syst. Technol. Manag. 3 (2006) 2–18, http://dx.doi.org/10.1590/S1807-17752006000100002.
[2] T.R. Gruber, A translation approach to portable ontology specifications, Knowl. Acquis. 5 (1993) 199–220, http://dx.doi.org/10.1006/knac.1993.1008.
[3] D.L. McGuinness, F. van Harmelen, OWL Web Ontology Language, 2004.
[4] A. Gyrard, C. Bonnet, K. Boudaoud, SWoT: Semantic Web of Things, 2012.
[5] A. Wróblewska, T. Podsiadł y Marczykowska, R. Bembenik, G. Protaziuk, H. Rybiński, Methods and Tools for Ontology Building, Learning and Integration – Application in the SYNAT Project, 2012, pp. 121–151, http://dx.doi.org/10.1007/978-3-642-24809-2_9.
[6] S.H. Venu, V. Mohan, K. Urkalan, G. T.V., Unsupervised Domain Ontology Learning from Text, 2017, pp. 132–143, http://dx.doi.org/10.1007/978-3-319-58130-9_13.
[7] W. Gao, J.L.G. Guirao, B. Basavanagoud, J. Wu, Partial multi-dividing ontology learning algorithm, Inf. Sci. (Ny) 467 (2018) 35–58, http://dx.doi.org/10.1016/j.ins.2018.07.049.
[8] J. Wu, X. Yu, L. Zhu, W. Gao, Leave-two-out stability of ontology learning algorithm, Chaos Solitons Fractals 89 (2016) 322–327, http://dx.doi.org/10.1016/j.chaos.2015.12.013.
[9] A. Konys, Knowledge systematization for ontology learning methods, Procedia Comput. Sci. 126 (2018) 2194–2207, http://dx.doi.org/10.1016/j.procs.2018.07.229.
[10] T. Wang, W. Li, F. Liu, J. Hua, Sprinkled semantic diffusion kernel for word sense disambiguation, Eng. Appl. Artif. Intell. 64 (2017) 43–51, http://dx.doi.org/10.1016/j.engappai.2017.05.010.
[11] S. Tonelli, C. Giuliano, K. Tymoshenko, Wikipedia-based WSD for multilingual frame annotation, Artificial Intelligence 194 (2013) 203–221, http://dx.doi.org/10.1016/j.artint.2012.06.002.
[12] D.L. Pennell, Y. Liu, Normalization of informal text, Comput. Speech Lang. 28 (2014) 256–277, http://dx.doi.org/10.1016/j.csl.2013.07.001.
[13] W. Li, K. Guo, Y. Shi, L. Zhu, Y. Zheng, DWWP: Domain-specific new words detection and word propagation system for sentiment analysis in the tourism domain, Knowl.-Based Syst. 146 (2018) 203–214, http://dx.doi.org/10.1016/j.knosys.2018.02.004.
[14] M.N. Asim, M. Wasim, M.U.G. Khan, W. Mahmood, H.M. Abbasi, A survey of ontology learning techniques and applications, Database 2018 (2018) http://dx.doi.org/10.1093/database/bay101.
[15] N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, D. Tapias, Flexible ontology population from text: The OwlExporter, in: Proc. Int. Conf. Lang. Resour. Eval. {LREC}, European Language Resources Association, Valleta - Malta, 2010..
[16] K. Bontcheva, V. Tablan, D. Maynard, H. Cunningham, Evolving GATE to meet new challenges in language engineering, Nat. Lang. Eng. 10 (2004) 349–373.
[17] H. Cunningham, V. Tablan, A. Roberts, K. Bontcheva, Getting more out of biomedical documents with GATE's full lifecycle open source text analytics, PLoS Comput. Biol. 9 (2013) e1002854, http://dx.doi.org/10.1371/journal.pcbi.1002854.
[18] R. Witte, B. Sateli, The LODeXporter: Flexible generation of linked open data triples from NLP frameworks for automatic knowledge base construction, in: Proc. Elev. Int. Conf. Lang. Resour. Eval., European Languages Resources Association (ELRA), Miyazaki, Japan, 2018.
[19] T. Thongkrau, P. LalitroJwong, OntoPop: An ontology population system for the semantic web, IEICE Trans. Inf. Syst. E95-D (2012) 921–931, http://dx.doi.org/10.1587/transinf.E95.D.921.
[20] P. Cimiano, J. Völker, Text2onto, in: Int. Conf. Appl. Nat. Lang. to Inf. Syst., 2005, pp. 227–238, http://dx.doi.org/10.1007/11428817_21.
[21] M. Sintek, XML Tab, 2007.
[22] M.A. Musen, The protégé project, AI Matters 1 (2015) 4–12, http://dx.doi.org/10.1145/2757001.2757003.
[23] A. Gómez-Pérez, Ontology evaluation, in: Handb. Ontol., Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 251–273, http://dx.doi.org/10.1007/978-3-540-24750-0_13.
[24] M. McDaniel, V.C. Storey, Evaluating domain ontologies, ACM Comput. Surv. 52 (2019) 1–44, http://dx.doi.org/10.1145/3329124.
[25] J. Brank, M. Grobelnik, D. Mladenić, A survey of ontology evaluation techniques, in: Proc. 8th Int. Multi-Conf. Inf. Soc., 2005, pp. 166–169.
[26] A. Gomez-Perez, Some ideas and examples to evaluate ontologies, in: Proc. 11th Conf. Artif. Intell. Appl., IEEE Comput. Soc. Press, 1995, pp. 299–305, http://dx.doi.org/10.1109/CAIA.1995.378808.
[27] J. Bandeira, I.I. Bittencourt, P. Espinheira, S. Isotani, FOCA: A methodology for ontology evaluation, 2016, http://arxiv.org/abs/1612.03353.
[28] A. Duque-Ramos, J.T. Fernández-Breis, M. Iniesta, M. Dumontier, M. Egaña Aranguren, S. Schulz, N. Aussenac-Gilles, R. Stevens, Evaluation of the OQuaRE framework for ontology quality, Expert Syst. Appl. 40 (2013) 2696–2703, http://dx.doi.org/10.1016/j.eswa.2012.11.004.
[29] J. Bœgh, A new standard for quality requirements, IEEE Softw. 25 (2008) 57–63, http://dx.doi.org/10.1109/MS.2008.30.
[30] A. Bachir Bouiadjra, S.-M. Benslimane, FOEval: Full ontology evaluation, in: 2011 7th Int. Conf. Nat. Lang. Process. Knowl. Eng., IEEE, 2011, pp. 464–468, http://dx.doi.org/10.1109/NLPKE.2011.6138244.
[31] S. Tartir, I. Arpinar, M. Moore, A. Sheth, B. Aleman-Meza, OntoQA: Metric-Based Ontology Quality Analysis, 2005.
[32] S. Tartir, I.B. Arpinar, Ontology evaluation and ranking using OntoQA, in: Int. Conf. Semant. Comput. (ICSC 2007), IEEE, 2007, pp. 185–192, http://dx.doi.org/10.1109/ICSC.2007.19.

[33] N. Guarino, C.A. Welty, An overview of ontoclean, in: Handb. Ontol., Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 201–220, http://dx.doi.org/10.1007/978-3-540-92673-3_9.

[34] M. Fernández-López, A. Gómez-Pérez, The integration of OntoClean in WebODE, in: EON, 2002.

[35] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, D. Poshyvanyk, When and why your code starts to smell bad, in: 2015 IEEE/ACM 37th IEEE Int. Conf. Softw. Eng., IEEE, 2015, pp. 403–414, http://dx.doi.org/10.1109/ICSE.2015.59.

[36] M. Poveda-Villalón, M.C. Suárez-Figueroa, A. Gomez-Perez, A double classification of common pitfalls in ontologies, 2019.

[37] M. Poveda-Villalón, A. Gómez-Pérez, M.C. Suárez-Figueroa, OOPS! (OntOlogy Pitfall scanner!), Int. J. Semant. Web Inf. Syst. 10 (2014) 7–34, http://dx.doi.org/10.4018/ijswis.2014040102.

[38] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, MA, USA, 1999.

[39] T. Mens, T. Tourwe, A survey of software refactoring, IEEE Trans. Softw. Eng. 30 (2004) 126–139, http://dx.doi.org/10.1109/TSE.2004.1265817.

[40] K. Muûlu, Y. Brun, R. Holmes, M.D. Ernst, D. Notkin, Speculative analysis of integrated development environment recommendations, in: Proc. ACM Int. Conf. Object Oriented Program. Syst. Lang. Appl., OOPSLA '12, ACM Press, New York, New York, USA, 2012, p. 669, http://dx.doi.org/10.1145/2384616.2384665.

[41] D. Hou, Studying the evolution of the Eclipse Java editor, in: Proc. 2007 OOPSLA Work. Eclipse Technol. Exch., Eclipse '07, ACM Press, New York, New York, USA, 2007, pp. 65–69, http://dx.doi.org/10.1145/1328279.1328293.

[42] T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, C. Parnin, Do developers read compiler error messages?, in: 2017 IEEE/ACM 39th Int. Conf. Softw. Eng., IEEE, 2017, pp. 575–585, http://dx.doi.org/10.1109/ICSE.2017.59.

[43] K. Muslu, Y. Brun, R. Holmes, M.D. Ernst, D. Notkin, Improving IDE recommendations by considering global implications of existing recommendations, in: 2012 34th Int. Conf. Softw. Eng., IEEE, 2012, pp. 1349–1352, http://dx.doi.org/10.1109/ICSE.2012.6227082.

[44] N. Tsantalis, T. Chaikalis, A. Chatzigeorgiou, JDeodorant: IDentification and removal of type-checking bad smells, in: 2008 12th Eur. Conf. Softw. Maint. Reengineering, IEEE, 2008, pp. 329–331, http://dx.doi.org/10.1109/CSMR.2008.4493342.

[45] C.N. Johnson, The benefits of PDCA, 35, 2002.

[46] L. Da Dalt, S. Callegaro, A. Mazzi, A. Scipioni, P. Lago, M.L. Chiozza, F. Zacchello, G. Perilongo, A model of quality assurance and quality improvement for post-graduate medical education in europe, Med. Teach. 32 (2010) e57–e64, http://dx.doi.org/10.3109/01421590903199734.

[47] M.J. Taylor, C. McNicholas, C. Nicolay, A. Darzi, D. Bell, J.E. Reed, Systematic review of the application of the plan–do–study–act method to improve quality in healthcare, BMJ Qual. Saf. 23 (2014) 290–298, http://dx.doi.org/10.1136/bmjqs-2013-001862.

[48] A.M. Kholif, D.S. Abou El Hassan, M.A. Khorshid, E.A. Elsherpieny, O.A. Olafadehan, Implementation of model for improvement (PDCA-cycle) in dairy laboratories, J. Food Saf. 38 (2018) e12451, http://dx.doi.org/10.1111/jfs.12451.

[49] S. Rita, K. Lakshmi, Mechanics of how to apply Deming's PDCA cycle to management education, SSRN Electron. J. (2009) http://dx.doi.org/10.2139/ssrn.1353763.

[50] P.F. Whetzel, N.F. Noy, N.H. Shah, P.R. Alexander, C. Nyulas, T. Tudorache, M. Musen, BioPortal: enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications, J. Biomed. Semant. 8 (2017) 1–22.

[51] C. Jonquet, A. Toulet, E. Arnaud, S. Aubin, E.D. Yeumo, V. Emonet, J. Graybeal, M.-A. Laporte, M.A. Musen, V. Pesce, P. Larmande, AgroPortal: A vocabulary and ontology repository for agronomy, Comput. Electron. Agric. 144 (2018) 126–143, http://dx.doi.org/10.1016/j.compag.2017.10.012.

[52] T. Mossakowski, K. Oliver, M. Codescu, Ontohub - a repository engine for heterogeneous ontologies and alignments, 2012.

[53] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L.J. Goldberg, K. Eilbeck, A. Ireland, C.J. Mungall, T.O.B.I. Consortium, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R.H. Scheuermann, N. Shah, P.L. Whetzel, S. Lewis, The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration, Nat. Biotechnol. 25 (2007) 1251.

[54] eBiquity Research Group, Swooble: Semantic web search, 2007.

[55] L. Li, I. Yevseyeva, V. Basto-Fernandes, H. Trautmann, N. Jing, M. Emmerich, Building and using an Ontology of Preference-Based Multiobjective Evolutionary Algorithms, 2017, pp. 406–421, http://dx.doi.org/10.1007/978-3-319-54157-0_28.

[56] A. Constantin, S. Pettifer, A. Voronkov, PDFX: Fully-automated PDF-to-XML conversion of scientific literature, in: Proc. 2013 ACM Symp. Doc. Eng., ACM, New York, NY, USA, 2013, pp. 177–180, http://dx.doi.org/10.1145/2494266.2494271.

[57] D. Tkaczyk, P. Szostek, M. Fedoryszak, P.J. Dendek, Ł. Bolikowski, CERMINE: automatic extraction of structured metadata from scientific literature, Int. J. Doc. Anal. Recognit. 18 (2015) 317–335, http://dx.doi.org/10.1007/s10032-015-0249-8.

[58] M. Liakata, C. Q, L.N. Soldatova, Semantic Annotation of Papers: Interface & Enrichment Tool (SAPIENT), in: Proc. {B}io{NLP} 2009 Work. Association for Computational Linguistics, Boulder, Colorado, 2009, pp. 193–200.

[59] I.G. Councill, C. Leegiles, ParsCit: An open-source CRF reference string parsing package, in: Proc. Lang. Resour. Eval. Conf. (LREC-2008), Marrakesh, 2008.

[60] C. Ramakrishnan, A. Patnia, E. Hovy, G.A.P.C. Burns, Layout-aware text extraction from full-text PDF of scientific articles, Source Code Biol. Med. 7 (2012) 7, http://dx.doi.org/10.1186/1751-0473-7-7.

[61] S. Mittal, Tools for ontology building from texts: Analysis and improvement of the results of Text2Onto, IOSR J. Comput. Eng. 11 (2013) 101–117, http://dx.doi.org/10.9790/0661-112101117.

[62] R. Haldar, D. Mukhopadhyay, Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach, 2011.

Gabriela R. Roldán-Molina, was born in Quito (Ecuador) in1991. She is a computer engineer graduated from the University of the Armed Forces Espe (Quito - Ecuador) in 2015. She completed her master's degree in mobile computing (Leiria — Portugal) in 2017. Recently she is studying a Ph.D. at the University of Vigo collaborating with the SING group. She has worked as a software developer in different technology companies. She currently works as a full stack developer in a company in the banking sector.

David Ruano-Ordás: He was born in Galicia (Spain) in 1985 and received his Ph.D. in Computer Science from the University of Vigo (Spain) in 2015. He is computer science engineer with high experience on Linux administration and software development under the ANSI/C standard. He collaborates as researcher with the SING group belonging to the University of Vigo. Regarding to the research experience he is mainly focused in the Artificial Intelligence area (automatic learning, or evolutionary algorithms) applied to spam filtering and drugs-discovery domain. Finally, he has participated in several national and regional research projects and has been co-author of several articles published in journals belonging recognized editorials such as Springer–Verlag or Elsevier. (http://www.drordas.info/).

Vitor Basto-Fernandes graduated in information systems in 1995 (including an internship at Ascom Tech AG — Switzerland), post-graduated in distributed systems in 1997 and got his Ph.D. on multimedia transport protocols in 2006, all from University of Minho (Portugal), where he has also been teaching assistant. He worked at Integral Vision Inc (UK) as software quality engineer in 1996 and was co-founder of PSI-Information Systems Lda (Portugal) in 1997, where he was project manager in B2B e-commerce web-based software development. From 2005 he has been lecturing at the University of Tras-os-Montes e Alto Douro (Portugal), and invited assistant professor in the same university in 2007 and 2008. In 2008 he joined the Informatics Engineering Department of Polytechnic Institute of Leiria (Portugal) as adjunct professor, where he has been coordinator professor between 2014 and 2016. He coordinated the M.Sc. Program in Mobile Computing from 2010 to 2012, was head of the Research Center in Computer Science and Communications at Polytechnic Institute of Leiria between 2012 and 2016, and researcher in several international projects in the areas of information systems integration, anti-spam filtering and multiobjective optimization, and principal investigator of two funded FCT (the Portuguese national funding agency for science, research and technology) research projects in the areas of research management and research internationalization. He publishes regularly in top-tier journals and conference papers, and organized international events in the areas of his research interests, multiobjective optimization, information security and semantic web.

José R. Méndez was born in Galicia (Spain) in 1977. Currently, he is an associate professor belonging to the computer science department of University of Vigo. He worked as a system administrator, software developer, and IT (Information Technology) consultant in civil services and industry during 10 years. He is an active researcher belonging to SING group and, although collaborates in different applications machine learning, his main interests are the development and improvement of anti-spam filters. (http://moncho.mdez-reboredo.info/).