x2OMSAC - an Ontology Population Framework for the Ontology of Microservices Architecture Concepts

Gabriel Morais[1], Mehdi Adda[1], Hiba Hadder[1], and Dominik Bork[1,2]

[1] Université du Québec à Rimouski, 1595, boulevard Alphonse-Desjardins, Lévis (Québec), G6V 0A6, Canada,
gabrielglauber.morais, mehdi_adda, hiba.hadder @uqar.ca
[2] TU Wien, Business Informatics Group,
Favoritenstrasse 11, Vienna, Austria,
dominik.bork@tuwien.ac.at

Abstract. Applying the Ontology of Microservices Architecture Concepts (OMSAC) as a modelling language calls users to have expertise in ontology engineering. However, ontology practice remains restricted to a limited pool of practitioners, leading to a barrier to widely adopting such a modelling approach. Here, we present x2OMSAC, an ontology population framework that enhances the modelling of microservices architectures using OMSAC. We instantiate our framework by FOD2OMSAC, which limits modellers' manual tasks to data selection, cleaning, and validation of created models, thereby eliminating the need for ontology expertise and, consequently, expanding the potential of OMSAC adopters for modelling microservices architectures.

Keywords: OMSAC, ontology population, microservices, conceptual modelling, machine learning, OpenAPI, Docker-compose, feature modelling

1 Introduction

Microservices Architecture (MSA) is a recent software engineering paradigm based on a compositional approach. Systems built using this paradigm are arrangements of microservices providing limited functionalities, which are put together to form complex systems[6]. It has been widely adopted by industry[2] in different domains to address various challenges, from the modernization of monolithic financial applications to large IoT systems.

The Ontology of Microservices Architecture Concepts (OMSAC)[11] is the MSA domain ontology formalized in the Web Ontology Language (OWL2[18]) using the semantics of the Description Logic (DL). It supports modelling, exploration, understanding, and knowledge sharing of MSA concepts, and MSA-based systems representation. For instance, modellers can apply the OMSAC's terminology component (TBox) as a modelling language[12]. This application of OMSAC allows modellers to link heterogeneous concepts and viewpoints simultaneously, bringing the capability to explore MSAs holistically and produce

different system views according to stakeholders' information needs throughout semantic queries. The resulting models are instances of OMSAC's concepts, i.e., an assertion component (ABox) that, with the OMSAC's TBox, make up a knowledge base of microservices systems.

However, OMSAC – as any ontology – lacks an established process to handle *ontology population*, which is creating concept instances related to an exiting TBox, i.e., creating an ABox. The ontology population process comprises two essential components: The TBox and an instance extraction engine[15]. Manually executing this process requires "tremendous effort and time"[9] And calls for specialized expertise and specific ontology engineering and exploration tools. Consequently, having an information extraction system that automates this process is highly desirable[9].

Accordingly, this paper presents x2OMSAC an ontology population framework tailored for OMSAC. It aims to address the automation challenge by facilitating existing knowledge to automate the creation of OMSAC ABoxes. We also present FOD2OMSAC, an implementation of the x2OMSAC framework based on a semi-automatic approach which populates OMSAC knowledge bases from a restrained number of inputs: *F*eature models, *O*penAPI, and *D*ocker-compose files. This implementation applies Sentence Transformer (SBERT[16]), a Natural Language Processing (NLP) machine learning (ML) model, to linkage proposes. We evaluate x2OMSAC by using FOD2OMSAC on two open-source microservices systems. The source code and the evaluation kit are available at this paper's code companion repository[13].

The remainder of this paper is as follows. In section 2, we present x2OMSAC followed by FOD2OMSAC in section 3. In section 4, we present the background in SBERT and details concerning the semantic matching mechanism implemented in the framework instance. Then, we detail the adopted evaluation process (section 5). We discuss our framework in section 6 and close this article with a short conclusion and feature work perspectives in section 7.

2 x2OMSAC

We designed x2OMSAC inspired by the ontology population process presented in Petasis et al.[15], which comprises four components: Input (TBox and knowledge resources), instance extraction engine, population process, and validation (consistency check)[15,9]. Hence, x2OMSAC contains four steps: Knowledge resources selection, model extraction, creation, and validation.

Knowledge Resource Selection In the knowledge resources selection, one must identify appropriate resources containing knowledge about the microservice's concept to model. The resource may vary according to the perspective of the microservice system to be modelled. For instance, modelling microservices' functional perspective would require information sources containing knowledge related to the microservice's functionalities (e.g., requirements, conditions, constraints). In contrast, technical perspectives would call for non-functional infor-

mation sources. This knowledge should be cleaned and organized to be correctly used in the following steps.

Model Extraction and Creation The model's extraction and creation steps compose the x2OMSAC's instance extraction engine and ontology population. The selected knowledge sources are explored to identify instances of OMSAC concepts and their relations. The challenge here is linking instances representing different perspectives which may be extracted from distinct knowledge sources in various processing tasks or at other points in time. Indeed, these relations could be implicit and dictated by the modelling goal.

For instance, when modelling a given microservice, it is possible to describe the domain functionalities it implements and provide its specific implementation details. These are two perspectives in OMSAC that are explicitly linked at the TBox level, but doing the linkage at the instance level is challenging because it calls for pairwise analysis of instances' properties, and the knowledge resources selected could be unlikely to provide any clue to unveil them. Thus, one must implement mechanisms to identify these linkages regardless of the processed knowledge resource.

Therefore, the model creation step should handle the instance creation and linkage using the available information about existing instances in the ontology and the information extracted in the previous step as knowledge input.

Validation The validation step comprises the consistency check and linkage review tasks. The former should be conducted by assessing the created ABox consistency concerning the TBox rules, and the latter should verify that suitable linkages between different perspectives have been created. The linkage validation should focus on detecting instance linkages that are optional or hidden at the conceptual level, i.e., liking individuals from different perspectives.

3 FOD2OMSAC

This section presents FOD2OMSAC, which stands for *F*eature model, *O*penAPI and *D*ocker compose *to* OMSAC. It implements the x2OMSAC framework using a semi-automatic approach for extracting models covering three microservices' modelling perspectives: Functional, implementation and deployment. It uses three knowledge resources: Feature model descriptions to handle the functional perspective, OpenAPI (specification version 3) files for the implementation perspective, and docker-compose for the deployment perspective. We provide a detailed view of FOD2OMSAC in Figure 1.

FOD2OMSAC implements x2OMSAC's knowledge resource selection, extraction, creation and validation steps. It is a hybrid ontology population[9] approach using rule-based and ML-based techniques. It comprises manual tasks for knowledge resource selection and validation steps, and Python scripts implementing automatic instance extraction and creation steps. The automated tasks can be invoked by a script providing a command-line interface (CLI), the *micro_extractor*, which offers commands handling different inputs, allowing the user to create specific models using specific files. It is possible to create mod-

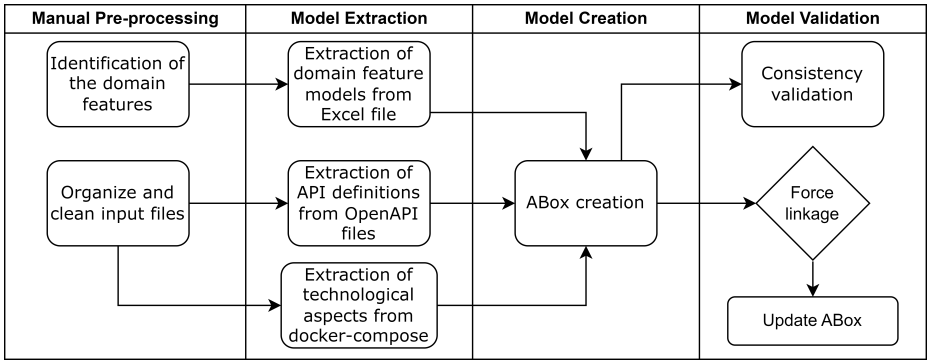


Fig. 1: FOD2OMSAC: Semi-automatic Approach for Creating OMSAC models

els from a unique system or create various systems simultaneously using batch mode. Figure 2 provides a screenshot of the CLI of the OMSAC ABox creator tool. In the following, we present each framework step in detail.

```
usage: micro_extractor.py [-h] -i INPUT (-fm | -sm) -n MODEL_NAME [-o OUTPUT_DIR]


Create an OMSAC-based model

options:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        The file (for functional domain models) or folder(s) (for
 systems models) from where the model will be extracted
  -fm, --functional_model
                        Creates a functional domain model
  -sm, --system_model   Creates a microservices system model
  -n MODEL_NAME, --model_name MODEL_NAME
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
```

Fig. 2: Command line interface of the OMSAC ABox creator tool.

3.1 Manual Pre-processing

The manual pre-processing prepares the different files to be processed by the *micro_extractor* scripts. The first task is defining the feature model input file, which must be a CSV file containing ten columns corresponding to concepts in the OMSC TBox. Table 1 provides the column names and meanings.

In the second task, the user verifies that the OpenAPI files are defined using version 3 of the OpenAPI specification. Otherwise, a conversion is needed and can be achieved manually using the Swagger Editor conversion facility.

Each microservice-based system must be organized in different folders if the batch mode is invoked. OpenAPI files must be named differently (e.g., the microservice name) to prevent overwriting.

Table 1: Description of the columns required by the CSV file to be processed.

Column	Content
goal	The goal's name, it is used as the goal ID.
goal_desc	A textual description of the goal.
requirement	The requirement's name, it is used as the requirement ID.
type	The requirement's type, it accepts functional or technical as values.
requirement_desc	A textual description of the requirement.
condition	The condition's name, it is used as the condition ID.
condition_desc	A textual description of the condition.
constraint	The constraint's name is used as the condition ID.
constraint_desc	A textual description of the constraint.
depends_on	The IDs of goals, requirements, conditions or constraints that have a dependency on each other.

3.2 Model Extraction

In this step, FOD2OMSAC extracts only mandatory and highly informative optional fields to guarantee higher compatibility. The order of extraction is fixed; thus, the user must start with files containing the feature models, then the OpenAPI files and finally, the Docker-compose file. When using the batch mode, these three files must be in each system folder.

When processing the OpenAPI files, the script first injects all the relative content to obtain an inline file. For doing so, it relies on the *dereference* function of the JSON Schema $Ref Parser [3], which enhances the extraction of the inputs and outputs of each endpoint. Then, the script extracts from each endpoint: The path, the operation, inputs and outputs, including input and output types and whether they are mandatory.

The extraction of information from the Docker-compose file identifies the type of services, including platform-provided services used by the microservices (e.g., database, messaging, API gateway, and aggregator), the Docker image used for each service and deployment dependencies.

Finally, the script merges all the information extracted from both files and stores the extracted models used in the model's creation step.

3.3 Model Creation

In this step, the script creates the OMSAC ABox instances and links them using the extracted data from the previous step. It relies on the OWLReady2[8] Python library and proceeds in the same order as the extraction step, starting with the creation of the functional perspective and then the implementation and deployment.

We also implemented the inter-perspective instance linkage mechanism. In other words, we link individuals from the OMSAC's *Feature* class to those of the OMSAC's *Requirement* class throughout the OMSAC's *fulfills* object property based on the instance names. It compares a short text similarity to a 0.7 threshold and automatically links individuals accordingly. The short text similarity is established using the *msmarco-distilbert-cos-v5* pre-trained SBERT model. The script generates a report of unmatched individuals (i.e., unmatched_features.csv) to be handled manually in the validation step.

[3] https://apitools.dev/json-schema-ref-parsert/

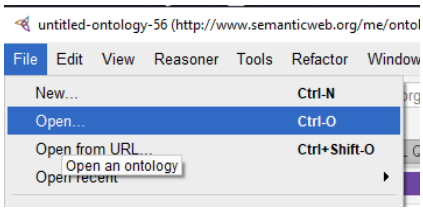
We detailed in Section 4 the process of the SBERT model's choice and the identification of the suitable threshold and provided background about SBERT NPL models.

3.4 Model Validation

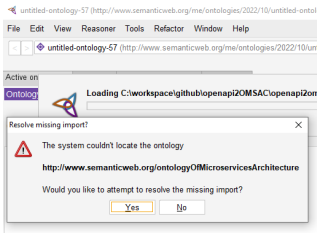
In this step, we validate the created models (i.e., the OMSAC's ABoxes). First, the user reviews the unmatched individuals and makes the necessary changes for complete linkage. Then, she validates the model consistency using Protégé[14].

To accomplish the first task, the user relies on the provided report of unmatched features (*unmatched_features.csv*), which contains two columns: Feature and requirement. The first column contains the Feature class individuals' names, and the second column is the name of the closest individual of the Requirement class that has been found. The user can then confirm the linkage, change it, or delete the line. Once the report is reviewed, the user can submit the modified report using the *force_linkage* command that will force the linkage.

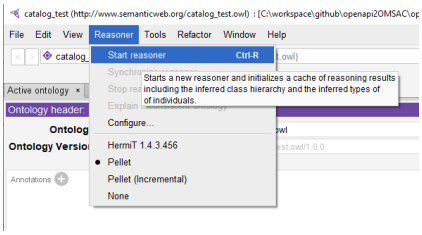
To assess the ABox's consistency, first, the user must open the ontology in Protégé (Figure 3a). A pop-up appears asking her to provide the path to the OMSAC TBox (Figure 3b). Finally, the user must start one of Protégé's provided reasoners, such as Pellet (Figure 3c). If the ABox is inconsistent, the reasoner will show an error message, and the user can ask Protégé to explain the causes of the inconsistency. In such a case, the user should identify the cause of the inconsistency and correct it manually.



(a) Opening Generated ABox



(b) Importing OMSAC TBox



(c) Starting Pellet Reasoner

Fig. 3: Manual ABox Consistency Validation with Protégé

4 The Semantic Comparison Mechanism

Using a word or character-based similarity measure may not consider that different words could represent a similar concept. NPL techniques have coped with this issue[3], and it seemed natural for us to rely on one of them for handling text comparison in FOD2OMSAC. This makes NPL machine learning techniques highly suitable for handling semantic similarity in heterogeneous knowledge source contexts[3]. Among these techniques, Bidirectional Encoder Representations from Transformers (BERT)[5] has become a state-of-the-art technique[3].

Sentence-BERT (SBERT)[16] are ML models applied for the NLP of short texts. SBERT fine-tunes BERT pre-trained models using siamese and triplet network architectures to "derive semantically meaningful sentence embeddings that can be compared using cosine-similarity."[16] SBERT has exceeded the performance of previous BERT models for establishing semantic similarity[7]. In addition, they need little effort to implement, as we can rely on a Python framework and a set of pre-trained SBERT ML models [4].

There is no ML model explicitly trained to process API descriptions based on the OpenAPI specification and able to match them to other natural language sources. Training such a model is out of the scope of this paper. Consequently, we decided to rely on existing pre-trained SBERT models used in the semantic search for automating concept linkages when building OMSAC ABoxes. Specifically, for linking the functional and implementation perspectives, i.e., Feature and Requirement individuals, extracted from the OpenAPI and the feature model files, respectively.

4.1 Selection of the SBERT Pre-trained Machine Learning Model

Our objective is to find a pre-trained SBERT ML model that minimizes manual actions in the OMSAC models' validation step and creates the fewest incorrect links. In addition, we need to couple the selected model to a mechanism to intercept unsuitable linkages that would be handled manually. We rely on a threshold mechanism to identify potential mismatches before effectuating the linkage so that they could be automatically placed for manual processing. Thus, we aim to identify which threshold most limits this risk. Consequently, we assessed selected pre-trained SBERT models considering as criteria the number of required manual processing, the number of incorrect undetectable linkages (i.e., mismatches with a similarity rate above the threshold) and their accuracy (using the accuracy and F1 measures).

First, we built an evaluation dataset by extracting 47 API endpoints from three microservices-based systems proposed by Assunçao et al.[1]: *EshopOnContainers*, *Hipstershop*, and *Socksshop*, and manually mapping them to a requirement extracted from the feature model of the e-shopping domain[10]. This evaluation dataset contains the expected matches the different pre-trained SBERT models should find.

[4] https://www.sbert.net/

Table 2: Assessment of the Pre-trained SBERT Models

Criteria	Required Manual Actions						Incorrect Undetected Link						Accuracy						F1					
Model /Threshold	**0**	**0.5**	**0.6**	**0.7**	**0.75**	**0.8**	**0**	**0.5**	**0.6**	**0.7**	**0.75**	**0.8**	**0**	**0.5**	**0.6**	**0.7**	**0.75**	**0.8**	**0**	**0.5**	**0.6**	**0.7**	**0.75**	**0.8**
msmarco-MiniLM-L6-v3	23	23	24	22	29	31	23	18	14	8	6	**1**	0.511	0.574	0.617	0.723	0.617	0.660	0.676	0.688	0.690	0.745	0.571	0.529
multi-qa-mpnet-base-cos-v1	**19**	**19**	23	26	28	32	19	12	8	6	2	**1**	0.596	0.660	0.617	0.596	0.617	0.553	0.747	**0.750**	0.667	0.612	0.571	0.432
msmarco-distilbert-cos-v5	**19**	**19**	20	23	30	31	19	15	9	3	2	2	0.596	0.617	0.681	0.702	0.574	0.553	0.747	0.735	0.746	0.708	0.500	0.462
msmarco-MiniLM-L12-cos-v5	24	23	24	25	26	30	24	19	15	11	4	2	0.489	0.574	0.617	0.638	**0.745**	0.681	0.657	0.688	0.690	0.667	0.714	0.571
multi-qa-MiniLM-L6-cos-v1	21	20	23	24	26	29	21	16	13	5	5	**1**	0.553	0.617	0.596	0.681	0.638	0.660	0.712	0.727	0.678	0.681	0.622	0.579
multi-qa-distilbert-cos-v1	21	20	20	24	25	29	21	14	9	6	2	2	0.553	0.638	0.702	0.660	0.723	0.638	0.712	0.730	**0.750**	0.667	0.698	0.564

Table 3: Selection of the Pre-trained SBERT Models

Model	**Threshold**	**Accuracy**	**F1**	**Required Manual Action**	**Undetected Incorrect Links**	**Criteria Met**
msmarco-distilbert-cos-v5	0.7	0.70	0.71	23	3	**2/4**
multi-qa-distilbert-cos-v1	0.75	0.72	0.70	25	2	0/4
multi-qa-MiniLM-L6-cos-v1	0.8	0.66	0.58	29	1	1/4

Following, we pre-selected six SBERT pre-trained models according to recent measurements of their performance rate in handling semantic search in asynchronous contexts[17,4]. We assessed their similarity score regarding our evaluation criteria using six thresholds: 0, 0.50, 0.60, 0.70, 0.75 and 0.80 (see Table 2 for the results).

To select the appropriate pre-trained SBERT model, we first identified the model and threshold pairs with the fewest undetected incorrect links and the smallest number of required manual actions. The pair multi-qa-MiniLM-L6-cos-v1 and 0.80 had the best scores (one undetected incorrect link and 29 required manual actions). Then, we recursively selected the next best score for undetected incorrect links having fewer manual actions than the previously selected models and kept the best pair model threshold. Finally, we looked at each pair and their performance regarding the four criteria announced above and kept the combinations that met the most criteria. Table 3 summarises the set considered and the selection rounds. From this analysis, we identified the *msmarco-distilbert-cos-v5* pre-trained SBERT model and the threshold of 0.7 as the most appropriate for our purpose.

5 Evaluation

We evaluated x2OMSAC's capacity to enhance the use of OMSAC as a modelling language. As it is an abstraction, we assessed its capabilities by evaluating the FOD2OMSAC implementation. For doing so, we relied on a comparison of the ABox generated by a junior developer with no ontology engineering knowledge using FOD2OMSAC to one developed manually by an ontology practitioner with three years of experience with ontology engineering and more than ten years of experience in modelling data structures (e.g., relational and no-relation databases) supported by a domain expert.

We relied on two open-source[1] microservices-based systems from the e-shopping domain as use cases: *EshopOnContainers* and *Socksshop*, and on the

e-shopping domain feature model proposed by Mendonça et al.[10]. The domain expert reviewed the proposed feature model and conducted the input files' organization and cleaning. This input was then used by both manual and automatic ABox creation. Table 4 summarizes the evaluation results of each model covering a particular perspective.

Table 4: Accuracy of FOD2OMSAC generated and manually created models.

Model	**FOD2OMSAC Accuracy**		
	Systems		*Average*
	eShopOnConatiners	*Sockshop*	
Functional model	100%	100%	100%
Implementation model	100%	100%	100%
Deployment model	53%	67%	60%
Inter-perspective linkage	67%	67%	67%

The evaluation results showed that the functional and implementation models created by FOD2OMSAC were similar to those made by the ontology practitioner. Concerning the inter-perspective linkage, we observed that both approaches had the same output for the Socksshop system and a difference of three links for the eShopOnContainers system. Nevertheless, FOD2OMSAC performed inaccurately when extracting deployment concept instances.

Regardless, when comparing the performance of FOD2OMSAC concerning the amount of data correctly processed, we observed that the distance between it and the manual approach is reduced. Indeed, the implementation perspective was built from seven OpenAPI JSON files representing more than three thousand lines of code. In contrast, the deployment perspective, which performed the worst, was built from two Docker-compose files representing 340 lines of code, ten times fewer lines.

6 Discussion

We understand that FOD2OMSAC improved the OMSAC ABox creation dramatically and validated the aim of the x2OMSAC framework for reducing the complexity of creating OMSAC models. The x2OMSAC met the need for technological openness mandatory to handle technological heterogeneity observed in the Microservices domain. Indeed, the implementation of the x2OMSAC presented in this paper handled only a limited set of technologies used for representing aspects of microservices-based architectures. By enabling FOD2OMSAC to process widely adopted standards, we ensured it covers many microservice-based systems, building a heuristic solution for creating OMSAC ABoxes that avoids manual-intensive tasks and limits the need for users' ontology knowledge. As observed by the user of the FOD2OMSAC, the CLI provided the information needed to pass through the extraction and creation steps, even if the user had no previous experience with Protégé.

We noted that the instance extraction from Docker-composed files performed poorly because the script covered a limited number of kinds of services, which were extracted based on comparing the service name and a set of words. For instance, if a service contained the string *data* or *db*, the script identified it as a *Database* instance. However, these files contain limited information, and the gap between potential and actual instances extracted could be filled by other knowledge sources, i.e., the OpenAPI files.

This work comes with limitations. The most obvious is the limited number of inputs used in the evaluation. We limited the experiment to an amount of information meeting human cognitive capabilities. Indeed, a larger set of inputs would make it impossible to compare the output of FOD2OMSAC to models created manually. Besides, the quality of the input used can impact the extraction accuracy and, consequently, the quality of the final models. Indeed, using resources extracted from source-code repositories may be prone to error because we have no guarantee that they are accurate and follow the standards of the state of the practice.

Similarly, the performance of the pre-trained S-BERT ML models we applied could impact the quality of the inter-perspective linkage and create undetectable unsuitable links when used in a different dataset. To handle this risk, we applied a threshold mechanism that can be tuned to ensure a higher intercept of undetectable mismatches. However, we expect the accuracy of the inter-perspective linkage mechanism to decrease if the input document uses a language other than English because, in such a case, the document will contain English (i.e. OpenAPI and docker-compose keywords) and non-English words likely to impact the semantic similarity performance, as the selected ML model was not trained with a multi-language dataset.

7 Conclusion

This paper presented the x2OMSAC, an ontology population framework tailored for the OMSAC ontology. We demonstrate how implementing this framework (FOD2OMSAC) based on a semi-automatic approach could handle existing knowledge sources to build models representing microservices architectures, which limits manual tasks, abstracts ontology engineering complexity for non-ontology practitioners, and effectively enhances the creation of OMSAC ABoxes. In future work, we plan to explore other machine learning models applied to NPL to increase the linkage mechanism' accuracy and expand its capabilities. Besides, we ambition to implement instances of x2OMSAC to handle other formalisms used in the microservices domain, including Proto Buffers and Kubernetes files.

Acknowledgements. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) grant number 06351, and Desjardins.

References

1. Assunção, W.K., Krüger, J., Mendonça, W.D.: Variability management meets microservices: six challenges of re-engineering microservice-based webshops. In: Proceedings of the SPLC (A). pp. 22.1–22.6 (2020)
2. Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A.: Microservices in industry: insights into technologies, characteristics, and software quality. In: IEEE International Conference on Software Architecture Companion. pp. 187–195 (2019)
3. Chandrasekaran, D., Mago, V.: Evolution of semantic similarity—a survey. ACM Computing Surveys (CSUR) 54(2), 1–37 (2021)
4. Craswell, N., Mitra, B., Yilmaz, E., Campos, D., Voorhees, E.M.: Overview of the trec 2019 deep learning track. arXiv preprint arXiv:2003.07820 (2020)
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
6. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. In: Present and ulterior software engineering, pp. 195–216. Springer (2017)
7. Han, M., Zhang, X., Yuan, X., Jiang, J., Yun, W., Gao, C.: A survey on the techniques, applications, and performance of short text semantic similarity. Concurrency and Computation: Practice and Experience 33(5), e5971 (2021)
8. Lamy, J.B.: Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. Artificial intelligence in medicine 80, 11–28 (2017)
9. Lubani, M., Noah, S.A.M., Mahmud, R.: Ontology population: Approaches and design aspects. Journal of Information Science 45(4), 502–515 (2019)
10. Mendonça, W.D., Assunção, W.K., Estanislau, L.V., Vergilio, S.R., Garcia, A.: Towards a microservices-based product line with multi-objective evolutionary algorithms. In: 2020 IEEE Congress on Evolutionary Computation. pp. 1–8 (2020)
11. Morais, G., Adda, M.: Omsac-ontology of microservices architecture concepts. In: 2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). pp. 0293–0301. IEEE (2020)
12. Morais, G., Bork, D., Adda, M.: Towards an ontology-driven approach to model and analyze microservices architectures. In: Proceedings of the 13th International Conference on Management of Digital EcoSystems. pp. 79–86 (2021)
13. Morais, G., Bork, D., Adda, M., Hadder, H.: Companion source code repository. https://github.com/UQAR-TUW/fod2OMSAC (2022)
14. Musen, M.A.: The protégé project: a look back and a look forward. AI Matters 1(4), 4–12 (2015), http://protege.stanford.edu/
15. Petasis, G., Karkaletsis, V., Paliouras, G., Krithara, A., Zavitsanos, E.: Ontology Population and Enrichment: State of the Art, pp. 134–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), https://doi.org/10.1007/978-3-642-20795-2_6
16. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019), http://arxiv.org/abs/1908.10084
17. Semantic BERT: Pretrained models, https://www.sbert.net/docs/pretrained_models.html
18. W3C OWL Working Group: Owl 2 web ontology language document overview (second edition) (2012), https://www.w3.org/TR/owl2-overview/