Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 184 (2021) 275–282

Procedia Computer Science

www.elsevier.com/locate/procedia

The 12th International Conference on Ambient Systems, Networks and Technologies (ANT)
March 23 - 26, 2021, Warsaw, Poland

CLONE: Collaborative Ontology Editor as a Service in the Cloud

Alexandros Preventis, Euripides G.M. Petrakis*

School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Crete, Greece

Abstract

The evolution of Web and cloud services technology has facilitated collaboration on the Web, providing the means for concurrent editing, change tracking and storing files in the cloud (e.g. Google Docs, Office 365). Ontology development teams could greatly benefit from this technology, that until now have been applied mainly to document processing. We introduce *CLONE*, a Web-based ontology editor that runs in the cloud and provides a real-time collaborative environment for creating and editing ontologies. *CLONE* is designed as a service-oriented architecture taking advantages of the easy extensibility and scalability features of this approach. *CLONE* provides all the essential features of stand-alone ontology editors, as well as significant collaboration features, including concurrent editing, editing history, team conversations and role-based access-authorization mechanisms.

Keywords: Ontology; Collaborative editor; Service Oriented Architecture; Cloud Computing

1. Introduction

Ontology engineering (e.g. creation, editing) can become a particularly tedious task especially for large ontologies in complex (e.g. scientific) domains [1, 2]. Building an ontology usually requires more than one skilled editor who is familiar with the ontology engineering processes and W3C specifications such as RDF(S) and OWL. This relates mostly with editing the structure of an ontology (e.g. adding new concepts or individuals) and enhancing ontology content with semantics (e.g. adding object properties). In fact, ontology engineering is a collaborative task in which, apart from developers, more participants can provide knowledge about the domain of interest. Braun et al. [3] describe this process as a informal learning process, referred to as "Ontology Maturing", that is carried out by collaboration. To enable collaboration, users participating in the ontology editing process need to have access to the ontology and be in constant communication with each other (e.g. for resolving differences in the understanding of ontology concepts, for committing changes and, for keeping track of the ontology versions).

Collaborative ontology editors facilitate manual ontology development by enabling groups of users to communicate and actively participate in the process. However, the volume of data that is being published in ontologies on the Web

* Corresponding author. Tel.: +030-28210-37229 ; fax: +030-28210-37542.
E-mail address: petrakis@intelligence.tuc.gr

1877-0509
10.1016/j.procs.2021.04.006

is huge so that, it is practically infeasible to be managed by people alone. Incorporating tools that integrate human and computational intelligence is a natural next step. Ontology engineering methods orient from a tool-assisted (but primarily manual engineering process) to a data-driven approach, in which human involvement is optimally leveraged for the resolution of issues that cannot be automatized.

We introduce *CLONE* (CLoud Ontology Editor), a lightweight, Web-based ontology editor that provides a real-time collaborative environment for creating and editing RDF(S) and OWL ontologies. It provides a user friendly and easy-to-use Web interface allowing users to create ontologies without being familiar with the peculiarities (e.g. syntax) of the underlying ontology language. *CLONE* runs in the cloud and requires no software installation on users' machines. Users can access and edit their ontologies from any place using a Web-browser.

CLONE is designed as a Service-Oriented Architecture (SOA) [4]. It comprises autonomous and reusable components (services) running on the same or different Virtual Machines (VMs) in the cloud and communicating with each other through RESTful interfaces [5]. The selection of SOA design is driven by the key requirements of today's systems for adaptability, low-cost maintenance and scalability. *CLONE's* architecture is modular and expandable and allows to balance (i.e. distribute) the work-load between the running VMs or, if necessary, to assign more resources to services which are stressed with many requests [6] (i.e. by scaling VMs horizontally or vertically). More services can be added on demand or, any service can be replaced or moved to a different VM (in the same or in a different cloud) with minimum overhead (i.e. only the IP of the service will change). *CLONE* runs as a Software as a Service (SaaS) application on the Web[1] and has been tested on very large ontologies (e.g. the Vaccine ontology[2]); no usability or performance issue was reported.

State-of-the-art solutions to the problem of ontology engineering are mainly centralized and single-user systems[3]. Nonetheless, *CLONE* is not intended to compete with business productivity solutions (e.g. Protégé[4]) in terms of performance or usability but rather, to show how a cost effective, collaborative ontology engineering system can be designed and built in the cloud using well-established, open-source technologies (i.e. Java, PHP, HTML, JavaScript etc.) and principles of SOA design. *CLONE* provides almost all essential features of classic ontology editors such as:

- *Edit/Create:* Users are opted to (a) create a new ontology, (b) upload and edit an existing ontology or, (c) import an ontology from the Web.
- *Storage:* Each user maintains a private repository with (possibly) more than one versions of the ontology.
- *OWL 2.0:* It supports expressiveness using the full set of OWL 2 axioms[5].
- *Reasoning:* Ontologies can be checked for consistency and new relations can be inferred from existing ones using Pellet[6] reasoner or Fact[7].
- *Edit control* with flow control buttons allowing to "Undo/Redo" up to 50 changes.
- *Version control* which allows to change the working version of an ontology (e.g. create a new version or, restore the ontology to a previous stored version).

These features are combined with desirable features of collaborative ontology editors foreseen in [3]:

- *Team editing:* Ontology owners can invite other users to collaborate in the engineering process.
- *Access control:* Ontology owners (moderators) assign roles to other users namely viewer, editor or administrator.
- *Concurrent editing:* All users involved in the editing process, can make changes on the ontology at the same time. All changes made by one user are immediately visible and accessible by all members of the editing team.
- *Change tracking:* A history log keeps the most recent changes made on the ontology by all users. The rollback functionality allows to "undo" or "redo" up to 50 changes.

[1] http://www.intelligence.tuc.gr/clone/
[2] https://raw.githubusercontent.com/vaccineontology/VO/master/src/VO_merged.owl
[3] https://www.w3.org/wiki/Ontology_editors
[4] https://protege.stanford.edu
[5] https://www.w3.org/TR/owl2-overview/
[6] https://www.w3.org/2001/sw/wiki/Pellet
[7] https://www.w3.org/2001/sw/wiki/Fact

- *Team conversation:* A real-time conversation mechanism facilitates communication between the members of the editing team.

The rest of this paper is structured as follows. Related work in discussed in Sec. 2. Issues related to *CLONE's* design and architecture and common use cases are discussed in Sec. 3 and in Sec. 4 respectively. Implementation issues are discussed in Sec. 5 followed by conclusions and issues for future research in Sec. 6.

2. Related Work

Most collaborative ontology editors are extensions of single-user systems. Only a few systems are designed explicitly (i.e. from scratch) to support collaboration. The later approach led to new decentralized engineering approaches [7]. WebProtégé [8] is the most important representative of the first category. It provides a Web front-end (for each user) which, in turn, connects to the WebProtégé back-end. The back-end runs a Tomcat[8] Web server and is responsible for handling user requests. It supports collaborative features such as concurrent editing of an ontology by multiple Web-based clients, tracks changes on the ontology etc. WebProtégé back-end connects to a Protégé server which implements access to a central version of the ontology, storage, and ontology engineering functionality (e.g. editing, reasoning etc.). However, role management is insufficient (i.e. all users have exactly the same edit rights and changes made by any user are immediately visible to all other users). Reasoning is supported but, only on the central version of the ontology which is stored on the Protégé server. Configuration and customization of the User Interface per ontology and per user is a desirable feature of WebProtégé that is not supported by other systems.

OntoWiki[9] is rather a collaborative Semantic Wiki application than a full-fledged ontology editor. It allows users to navigate through RDF knowledge-bases and view Linked Data[10] in a Wiki-like form (i.e. each user can add or edit content). RDF content can be edited in-line on the pages that are displayed. Each page can roll-back to previous states. Reasoning and consistency checking can be applied by invoking the DL-Learner plug-in[11]. Similar to most Wiki-based ontology editors, OntoWiki trades part of the expressiveness of the ontology with ease-of-use, resulting (compared to typical ontology editors) to limited expressiveness. It does not provide role management (allows any user to modify the ontology at will) and does not provide a communication mechanism for the collaborating users.

NeOn Toolkit[12] provides a collaborative ontology environment which, apart from ontology editing, supports a variety of ontology engineering activities using different plug-ins for different functionality, including annotation and documentation, users interaction, ontology matching, reasoning etc. Concurrent ontology editing and central version control are not supported. Instead, every user works on a local copy of the ontology and needs to invoke the "OWLdiff plug-in" in order to detect changes and (if desired) apply the differences on the central copy of the ontology. Finally, NeOn Toolkit does not provide sufficient role management capabilities and any contributor can apply any changes on the ontology. The last version of Neon Toolking was announced in 2011.

An important feature of all systems (and also of *CLONE*) is collaborative editing of large ontologies and users interaction over the network. Some actions are performed locally (i.e. on the computer of each user) while others are performed remotely in the cloud or server. Existing applications promise high concurrency and real-time collaboration with minimal latency. However, not all desirable features (including those foreseen in [3]) are supported by all systems. Table 1 summarizes all desirable features and checks those which are supported by each individual system. Obviously, *CLONE* overshadows all its competitors and is the only system which is SOA, cloud-based and offers full-fledged ontology editing capabilities for OWL 2.0 ontologies (e.g. supports editing at class - structural, at instances level and, all OWL 2.0 axioms).

Security features are not supported by any system in Table 1. Securing the collaborative editing environment is a challenging task and has not been taken into account by existing systems. The principles of Security by Design and, Security and Privacy by Default [9, 10] must be applied since the design phase of a system. The system is exposed to risks due un-authorized attempts to access services. These can be handled successfully with the aid of traditional security methods (e.g. encryption, authorization, auditing). However, a system is also vulnerable due to malicious

[8] http://tomcat.apache.org
[9] http://ontowiki.net
[10] https://www.w3.org/standards/semanticweb/data
[11] https://dl-learner.org
[12] http://neon-toolkit.org/wiki/Main_Page.html

Table 1: Related work and contributions of *CLONE*.

System	OntoWiki	NeOn Toolkit	WebProtégé	CLONE
Concurrent editing	√		√	√
Edit roll-back	√	√	√	√
Versioning			√	√
Reasoning	√	√		√
Role management			√	√
Users interaction		√	√	√
Cloud				√
Service oriented				√
OWL 2.0 support			√	√
Security				
Customization			√	

users operating at the Web front-end. The security mechanisms must be complemented with trust evaluation methods for dealing with these risks. EU's GDPR[13] has a significant impact on systems design. Data protection is also crucial, as potential intrusion may not only lead to vulnerable personal data theft but may risk system operation overall.

3. Design and Architecture

We followed a valid design approach that identified functional and non-functional system requirements and specifically, (a) functional components and their interaction, (b) information that is managed and how it is acquired, transmitted, stored and analyzed, and (c) different types of users and how they interact with the system.

Each user belongs to a user class and has a role. Each user class is assigned a role encoding her/his authorization to access ontologies and services. Each user has an identifier, an email and a name being displayed. The following user groups and functional requirements associated with each group are identified:

- *Viewers:* they can view the ontology but they are not allowed to make changes. Viewers can also view the change history and participate in conversations with other users.
- *Editors:* editors can perform all actions that viewers can. Moreover, they can modify ontology entities. Notice, that customization of user access per project or user is not supported, so each editor has access to the entire ontology.
- *Administrators or ontology moderators or owners:* they configure, maintain and monitor the ontology engineering process. They are responsible for performing Create, Read, Update, Delete (CRUD) operations on (a) users (e.g. they can register new users to the system and assign them access rights) and, (b) ontologies (e.g. they can register new projects in order to start a new ontology process). They are responsible for monitoring users' activities. Ontology administrators have unrestricted access to the ontology. They can manage the ontology and can create new versions of the ontology. Ontology creators are automatically assigned the administrator role.

Depending on their role, users are entitled to perform either ontology management or editing operations (or both in the case of administrators). Ontology management relates to actions on ontologies (e.g. CRUD operations on files and users). New projects (i.e. ontologies) can be started by administrators. Activities include also uploading an existing ontology from a user's machine or the Web or, downloading an ontology on the user's disk. Administrator can also invite new users to participate in the engineering process and manage their roles (e.g. renounce any permissions granted to other users), change project name or status (e.g. finalize or delete a project). Administrators can create or delete (e.g. obsolete) ontology versions at any time. Administrators can change the active version of an ontology

[13] https://eugdpr.org

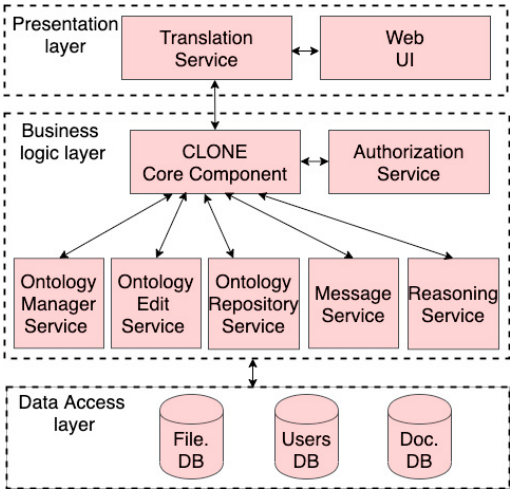

Fig. 1: Layered architecture.

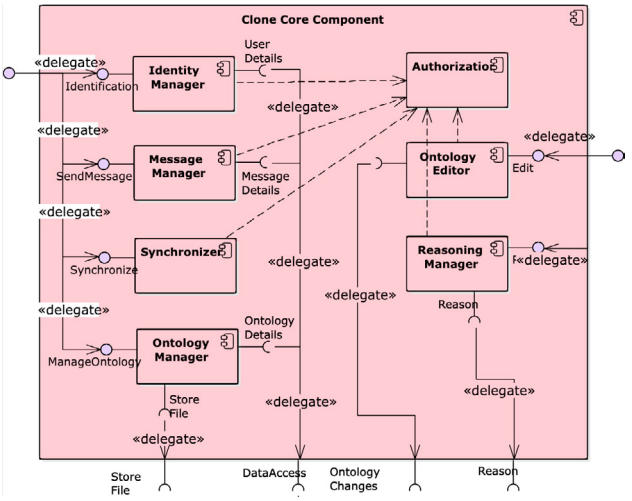

Fig. 2: Core component service.

or role-back to an earlier version. A version is a snapshot of the current state of the ontology that can serve as a restoration point.

CLONE adopts a 3-layer architecture design model. Fig. 1 illustrates *CLONE's* layered architecture. Details on the architecture design and its implementation can be found in the author's thesis [11].

Presentation layer implements the Web interface and a two-way communication between the user and the system: it forwards user commands to the *Business Logic layer* and (conversely), it is responsible for displaying data (e.g. ontology axioms) it receives from the *Business Logic layer* in human-readable form. Ontology editing commands as well as, ontology axioms in RDF(S) or OWL are encoded in JSON prior to transmission and (conversely) are decoded for viewing. This is a responsibility of the *Translation* service.

Business Logic layer orchestrates, controls and executes services in order. This is a responsibility of the *CLONE Core Component* service. When a request is received, it is dispatched to the appropriate service and forwards the responses to the Web front-end. Fig. 2 illustrates services within the *CLONE Core Component* service. Requests regarding user accounts and access rights are dispatched to user *Authorization* service. It implements access control to services based on user roles and access policies. Services in *CLONE* do not expose their interface to the Web without protection. The *Authorization* service performs basic security controls such as (a) checks user credentials prior to accessing the system (b) checks user access rights prior to authorizing a user to access a service (e.g. whether a user has the permission to access or edit the ontology depending on her/his role), (c) checks if a session between the user front-end and the system back-end has been has been established and is active. In cases of rule violations, appropriate notifications are generated and forwarded to the Web front-end. This is a responsibility of the *Message Manager* service. The same service handles the messages that are exchanged between users on a *chat* service using Server Sent Events (SSE).

An ontology session is initialized at login and remains active during a time interval which is also specified in advance. For each user, session information is collected by the *Messaging service* and stored in the *Users database* in the *Data Access* layer together with user profiles, messages and actions performed during a session.

Ontology Manager service is responsible for the management of ontology files and their versions. This is a rather complex service that implements functionality pertaining to creation and loading of ontology files in the main memory as well as, to the deletion and storage of ontology files on the disk. Once an ontology is loaded in the main memory, accessing its contents (OWL 2.0 axioms) is a responsibility of the *Ontology Editing Service* (OES). This operation is described in detail on Sec. 4. The *ontology collector* service periodically scans the main memory in order to free the space occupied by ontologies which haven't been accessed for the last 15 minutes. The *Synchronizer* consolidates changes made by the users using Server Sent Events (SSE).

The, *Reasoning Manager* service is responsible for invoking the reasoner (i.e. Pellet) and for forwarding reasoner's results to the Web front-end. When invoked by a user, it requires two URLs, one specifying the input ontology and

one specifying the output ontology. If the ontology is consistent, the output ontology is written to the location of the second URL. Moreover, the service will notify the user about the results.

Data Access Layer implements database functionality for (a) ontologies (b) users and, (c) metadata. Ontology files are stored in the *File DB* which is implemented as a simple disk directory. This is a responsibility of the *Ontology Manager* service. Additional (metadata) information associated with the ontology engineering process are, project identifier, resource location on the Web, user name and ontology owner, time and date the ontology was created or modified and, current version identifier. This information is stored in JSON format in the *Documents database* which is implemented as a Non-SQL database using MongoDB. Finally, users' profile information and users access rights, as well as messages and error messages generated are stored in the *Users database* using MysSQL.

4. Use Cases

Initially, users register to *CLONE* to receive a login name. This is a responsibility of the cloud administrator. Once a user is logged-in, she/he is prompted to select an ontology for editing. Only ontologies for which she/he has granted access by their owner are displayed. Alternatively, the user can become an ontology owner her/himself and start a new project (i.e. create a new ontology or load an existing ontology from a file or the Web). In this case, she/he is entitled to invite other users to participate in the ontology engineering process (as viewers or editors). User actions are categorized by functionality into *Ontology Management*, *Ontology Editing* and *Ontology Reasoning* actions.

Ontology Management implements the following actions on ontology files per user class (in parenthesis):

- *Create new project* (for owners): create a new (empty) ontology; load an ontology from a Web location or from local machine.
- *Edit project* (for owners): change project, ontology name and metadata.
- *Create new version or delete obsolete or erroneous versions of an ontology* (for owners).
- *Change the active version of an ontology* (for owners). Owners can swap between ontology versions(i.e. set new active version, roll-back to a previous version).
- *View ontology metadata* (for viewers, editors, owners).
- *Download ontology* (for viewers, editors, owners).
- *Leave the editing team of project* (for viewers, editors). By doing so, the owner renounces any permissions that they had been granted to a user leaving the project.
- *Manage user roles* (for owners). Ontology can invite other users with a role.

Ontology Editing relates to actions pertaining to the ontology editing process (i.e. actions that change the ontology contents). In the following scenario, a user (editor or administrator) logs into *CLONE* using her/his credentials. The user is prompted to select an ontology from the list of the ontologies she/he is entitled to edit. The ontology is retrieved from the *File database* in RDF/XML or, it is loaded from the *ontology data store* in the main memory (if it has been loaded in the main memory already by another user). The ontology is then serialized in JSON format and returned to the *Clone Core Component* which, in turn, converts the ontology relations to human readable form for display.

Assume that the ontology has been opened in OES. In the user interface environment, ontology entities are grouped in various tabs by type (i.e. classes, object properties, individuals, etc.). When the user selects an entity tab, the *Clone Application* responds with the respective entity. The user can click on the entity to view relations (i.e. axioms) associated with it. Any edit changes are forwarded to the *OES* which applies the changes to the ontology in two steps: (a) it sends a request to the *ontology repository* (on the disk) to store the changes in the current ontology version and, (b) it responds to the *Clone Application* with the changes applied on the ontology (on the disk). The *Clone Application* responds by displaying the updated axioms to all connected users.

Once the ontology has been opened in OES and the user clicks on the *Apply Reasoning* button the *Reasoning Service* retrieves the ontology from the *Ontology Editing Service* and invoked the reasoner. If the ontology is inconsistent, the *Clone Application* notifies the user on the result and the cause of error. If the ontology is consistent, the *Reasoning Service* forwards the inferred ontology to OES. The user may select to store the inferred ontology on the disk or, display it for viewing in a separate window.

5. Implementation

The system comprises autonomous RESTful services communicating with each other over HTTP. A service or group of services can be deployed in the same or in different Virtual Machines (VMs) in the cloud [11]. This is a highly flexible design that allows distribution of the workload to different VMs and asynchronous execution of long processes in the background. *CLONE* is deployed in 4 VMs. Each VM runs Ubuntu 18.04 and has a Web Server installed that allows a component to work independently as a Web Service and inter-operate with other VMs (i.e. services installed in VMs) using REST calls. *CLONE* is written in Java using the JAX-RS API[14] for implementing REST communication and OWL API[15] for implementing the software interface with OWL 2.0. Fig. 3 illustrates the services installed in each VM and their interconnection network.

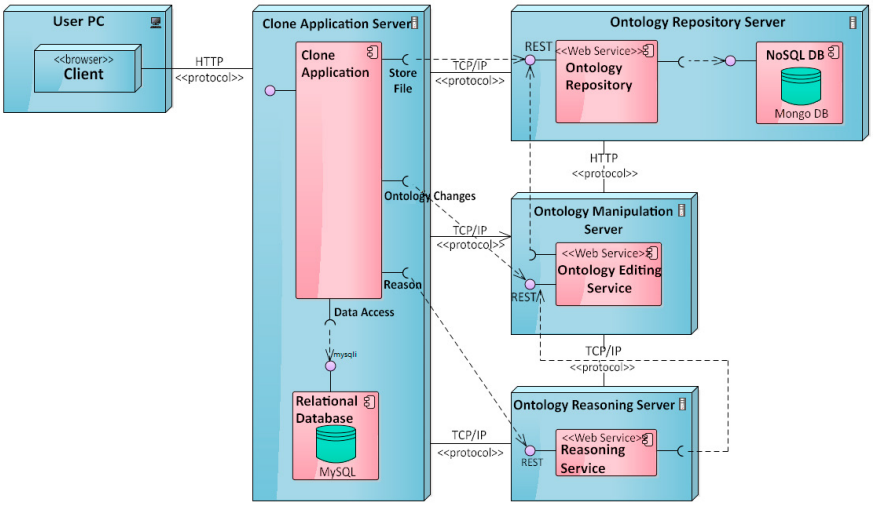

Fig. 3: *CLONE's* deployment in the cloud.

Once login, the *home page* presents the list of ontologies that the user is entitled to edit. The user is opted to create a new project (i.e. start a new ontology or load one from a file or the Web). In the later case, the user becomes owner of the ontology, invites other users to participate in the ontology engineering process and, assigns each user a role. A new user can be added to the ontology team by entering her/his username or e-mail and by selecting her/his role from a drop-down menu. Administrators are authorized to change the current active ontology version. When the active ontology version is changed, any users editing or viewing a previous version will lose access to it.

By opening an ontology for editing, OES displays the active version of the ontology. The user is opted to select the entities for editing (e.g. classes, object properties, data properties, individuals) by clicking on the corresponding button. The user is also opted to create new version, download the ontology on the disk or, invoke the reasoner. The *Ontology tab* provides access to ontology properties such as ontology URI, version, annotations (i.e. add short descriptions, names of editors etc.). *Classes tab* are presented in a tree view (i.e. each class appears below its parent). This tab allows users to browse through the classes, create new classes, edit or delete existing ones. In a similar way, data and object properties and also class individuals are displayed and can be edited. Object properties can receive additional characteristics and can be functional, inverse functional, reflexive, irreflexive, symmetric, asymmetric or transitive. Disjoint and equivalence properties can be also defined.

Data Properties tab allows users to manage data property definitions. It is identical to the *Object Properties tab* in all but, (a) the range of data properties is a datatype whereas in object properties is a class and, (b) data properties can have only the functional property. Individuals represent actual objects in the domain of interest. They can be *Named* individuals (i.e. they have been assigned an explicit name and a URI) in which case, they can be used by any ontology or, *Anonymous* individuals, in which case they don't have a global name in which case, they can be referenced only

[14] https://jax-rs.github.io/apidocs/2.0/
[15] https://www.w3.org/2001/sw/wiki/OWLAPI

within the current ontology. *CLONE* handles only *Named* individuals. The *Datatypes tab* includes a list with the built-in OWL datatypes, but users can create their own datatypes as well or, add annotations on datatypes.

A user can display (a) a history menu with "roll-back" functionality which allows a user to "undo" or "redo" up to 50 recent changes and (b) a conversation window which implements "chat" functionality which allows the user to communicate with other users.

6. Conclusions

CLONE is a Web-based full-fledged ontology editor supporting real-time collaborative editing of OWL 2.0 ontologies. It supports full expressiveness of OWL 2.0 according to the W3C specification and provides all the essential features of an ontology editor together with collaboration features, such as version management, cloud deployment, various access levels depending on user roles and concurrent editing. In the short term, we plan to evaluate *CLONE* by running a usability study based on questionnaires addressing various aspects of system functionality (e.g. performance, functionality, ease of use) for different user categories. *CLONE* is currently being extended to support customizable views for different users depending on their role and expertise, additional formats such as Manchester and Turtle and, incorporating a triple store database (e.g. Virtuoso[16]) for storing native OWL 2.0 relations and metadata. A future implementation of the user authentication and authorization mechanism will be based on oAuth2.0[17]. Additional, long term goals, include incorporating support for temporal and spatio-temportal ontologies in the example of Chronos-Ed [2]. Handling risks due to malicious behavior of users is still an open issue. Incorporating scalability features for dealing with increased workloads is an important direction for future work. HTTPS protocol will eventually replace HTTP as a secure solution for the transmission of confidential information over the public network.

References

[1] O. Corcho, M. Fernández-López, A. Gómez-Pérez, Ontological Engineering: Principles, Methods, Tools and Languages, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 1–48.
URL https://doi.org/10.1007/3-540-34518-3_1

[2] A. Preventis, E. G. M. Petrakis, S. Batsakis, Chronos ed: A tool for handling temporal ontologies in protégé, Intern. Journal on Artificial Intelligence Tools 23 (4) (2014).
URL https://www.worldscientific.com/doi/abs/10.1142/S0218213014600185

[3] S. Braun, A. Schmidt, A. Walter, G. Nagypal, V. Zacharias, Ontology maturing: a collaborative web 2.0 approach to ontology engineering, in: Proc.of Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th Intern. World Wide Web Conf. (WWW 2007), Banff, Canada, 2007.
URL http://publications.andreas.schmidt.name/ontology_maturing_braun_schmidt_walter_www07.pdf

[4] T. Erl, SOA Principles of Service Design, Prentice Hall, Upper Saddle River, NJ, USA, 2007.
URL https://dl.acm.org/citation.cfm?id=1296147

[5] S. Schreier, Modeling restful applications, in: ACM International Workshop on RESTful Design (WS-REST'11), Hyderabad, India, 2011, pp. 15–21. doi:10.1145/1967428.1967434.

[6] M. S. Alexiou, E. M. Petrakis, Elixir: An agent for supporting elasticity in docker swarm, in: Proc. of Advanced Information Networking and Applications (AINA 2020), Springer International Publishing, Caserta, Italy, 2020, pp. 1114–1125, aISC, Vol. 1151.
URL https://link.springer.com/chapter/10.1007%2F978-3-030-44041-1_96

[7] E. Simperl, M. Luczak-Rösch, Collaborative ontology engineering: A survey, The Knowledge Engineering Review 29 (1) (2013) 101–131.
URL https://pdfs.semanticscholar.org/8a60/e43dcd24abacc493fa10a76fb88f98d1d5bd.pdf

[8] T. Tudorache, C. Nyulas, N. Noy, M. Musen, Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web, Semantic Web Journal 4 (1) (2013) 89–99.

[9] A. Cavoukian, M. Dixon, Privacy and security by design: An enterprise architecture approach (Sep. 2013).
URL https://ww.ipc.on.ca/wp-content/uploads/Resources/pbd-privacy-and-security-by-design-oracle.pdf

[10] S. Sotiriadis, E. G. M. Petrakis, S. Covaci, P. Zampognaro, E. Georga, C. Thuemmler, An architecture for designing future internet (fi) applications in sensitive domains: Expressing the software to data paradigm by utilizing hybrid cloud technology, in: 13th IEEE International Conference on BioInformatics and BioEngineering, 2013, pp. 1–6.
URL https://ieeexplore.ieee.org/abstract/document/6701578

[11] A. Preventis, Clone: Cloud ontology editor, Tech. Rep. TR-TUC-ISL-02-2020, MSc Thesis, School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Crete (Oct. 2020).
URL https://dias.library.tuc.gr/view/87173

[16] https://virtuoso.openlinksw.com
[17] https://oauth.net/2/