OntoEditor: Real-time Collaboration via Distributed Version Control for Ontology Development

Ahmad Hemid[1], Waleed Shabbir[2], Abderrahmane Khiat[2], Christoph Lange[1,4], Christoph Quix[1,3], and Stefan Decker[1,4]

[1] Fraunhofer FIT, Data Science and Artificial Intelligence, Germany
{ahmad.hemid,christoph.lange-bever,christoph.quix,stefan.decker}@fit.fraunhofer.de
[2] Fraunhofer IAIS, Enterprise Information Systems, Germany
{waleed.shabbir,abderrahmane.khiat}@iais.fraunhofer.de
[3] Hochschule Niederrhein, Germany
christoph.quix@hs-niederrhein.de
[4] RWTH Aachen University, Germany
lange@cs.rwth-aachen.de,decker@dbis.rwth-aachen.de

Abstract. In today's remote work environment, the demand for real-time collaborative tools has surged. Our research targets efficient collaboration among knowledge engineers and domain experts in Ontology development. We developed a web-based tool for real-time collaboration, compatible with GitLab, GitHub, and Bitbucket. To tackle the challenge of concurrent modifications leading to potential inconsistencies, we integrated an Operational Transformation-based real-time database. This integration enables multiple users to concurrently collaborate to build and edit their ontologies, ensuring both consistency and atomicity. Furthermore, our tool enhances user experience by providing meaningful syntax error messages for ontologies expressed in various RDF serialization formats. This streamlined the manual correction process. Additionally, we established a reliable synchronization channel for users to allow pulling and committing changes to distributed repositories for their developed ontologies. Yielding promising results, our evaluation focused on two key aspects: first, assessing the tool's collaborative editing consistency via an automated typing script; second, conducting a comprehensive user study to evaluate its features and compare its functionalities with similar tools.

Keywords: Real-time collaboration · RDF serialization · Version Control Systems · Git integration · Error detection · Syntax validation · Ontology development.

1 Introduction

The vision of the Semantic Web, fostered by the World Wide Web Consortium (W3C), aims to transform the web into a machine-interpretable platform, akin to a global database [1]. However, even with all research efforts conducted to

achieve this goal, the current landscape of the web lacks an interconnected data framework, leading to fragmented data confined within individual applications.

At the heart of the Semantic Web lies the Resource Description Framework (RDF) [2], pivotal for modeling data and its relationships. Yet, contemporary applications grapple with challenges such as limited real-time collaboration and inadequate syntax validation within RDF-based systems, demanding an integrated and comprehensive solution.

This paper introduces OntoEditor, an innovative Online Collaborative Ontology Editor designed to revolutionize real-time collaboration across various RDF serialization formats. By harnessing Version Control Systems (VCS) such as GitHub, GitLab, and Bitbucket, OntoEditor offers users live syntax validation and collaborative editing features, effectively tackling inherent limitations in current systems, particularly in conflict resolution.

OntoEditor fills a critical gap in the domain of collaborative ontology development, offering a robust platform that seamlessly integrates real-time collaboration capabilities with thorough syntax validation across multiple RDF serialization formats. Through this endeavor, we aim to alleviate the persistent challenges faced by users in effectively collaborating on ontology projects.

This paper is structured as follows: Section 2 presents a motivational example, while Section 3 explores related works, offering a comparative analysis of existing solutions concerning OntoEditor. In Section 4, an overview of OntoEditor, including its workflow and features, is provided. Section 5 details its implementation and the technologies employed. The evaluation, conducted through experimental tasks and a user study, is presented in Section 6. Section 7 discusses the current limitations of OntoEditor, while Section 8 explores strategies for its sustainable adoption. Finally, the paper concludes in Section 9, with potential avenues for future work highlighted in Section 10.

2 Motivation

Suppose John, along with his colleagues Robert and Lisa, aims to develop an ontology together, illustrating the need for real-time collaboration as depicted in Figure 1. Their expertise in ontology engineering leads them to prefer plain text editors and VCS such as Git[1] for collaborative ontology development. However, existing tools lack efficient collaboration, communication, and real-time syntax error detection. This results in a cumbersome and error-prone process where users must separately write, check syntax, communicate changes, and repeat this cycle, causing inefficiency and errors.

Research from Queens University of Charlotte highlights that about 75% of employers highly value teamwork and collaboration, emphasizing the need for streamlined collaboration tools[2]. This inspired us to devise a solution that enables multiple users to collaborate seamlessly during ontology development without constant syntax-checking interruptions.

[1] `https://git-scm.com/`

[2] `https://blog.bit.ai/collaboration-statistics/`

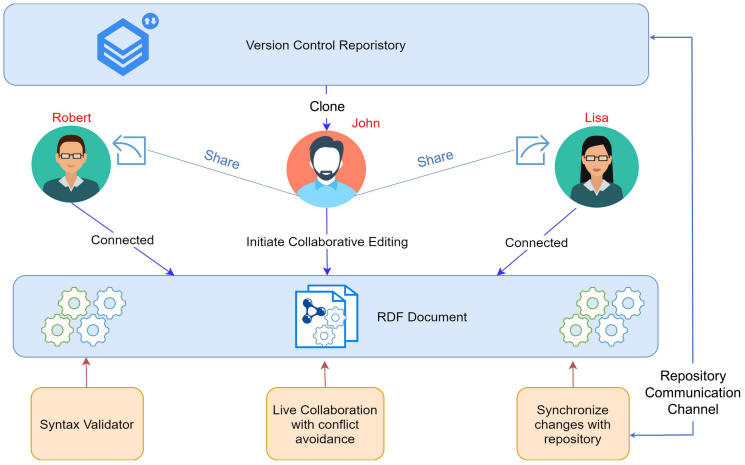

Fig. 1: **Collaboration Scenario.** An example motivating collaborative ontology development with a built-in syntax validator. Both Robert and Lisa seek to work seamlessly with John without requiring software downloads or installations.

Digging deeper into the scenario presented in Figure 1, John, Robert, and Lisa aim to collaborate on an ontology. They can create a new ontology or import an existing one from VCS such as GitHub. John initiates collaborative editing, sharing the link with his colleagues, enabling real-time simultaneous document viewing and tracking of each user's edits.

With the syntax checker enabled, any modifications trigger instant error detection visible to all users, fostering discussions and corrections in real-time via a shared chat. This approach ensures everyone's awareness of errors and each other's editing progress.

The final step involves users synchronizing their work with the remote repository. Only authorized users can commit changes, ensuring controlled access and ownership permissions within the repository.

3 Related Work

This section delves into collaborative ontology development, ontology synchronization with VCS, and tools supporting various RDF formats and validation.

3.1 Parsing RDF and Syntax Checking

Numerous online and desktop tools specialize in validating RDF data and supporting syntax checking. The W3C RDF validation tool [11] primarily focuses on parsing and validating RDF/XML[1] exclusively. Meanwhile, isSemantic RDF

[1] `https://www.w3.org/TR/rdf-syntax-grammar`

Tools [7] offer syntax checking and format conversions among other visualization and generation services but lack comprehensive support for collaborative editing or VCS integration. Our objective revolves around supporting ontology development within distributed VCS, a feature noticeably absent in existing tools.

Table 1: **Feature Comparison.** OntoEditor, TurtleEditor, and WebProtégé for Collaborative Ontology Development.

Feature	**Tool**		
	OntoEditor	**TurtleEditor**	**WebProtégé**
Real-time Collaboration	✓	×	✓
Textual RDF Editor	✓	✓	×
RDF Serializations	Turtle, RDF/XML, JSON-LD	Turtle	Turtle, RDF/XML
Integration with Git	✓ (GitHub, GitLab, Bitbucket)	✓ (GitLab only)	× (Own Protégé Server)
Conflict Resolution	✓	×	✓
Export/Download Option	✓	×	✓

3.2 Ontology Editors

Our exploration identified tools with collaborative capabilities but notable limitations. Existing tools often bound ontology development to specific formats or lack crucial collaboration features. Table 1 compares OntoEditor with similar tools such as TurtleEditor and WebProtégé. TurtleEditor [10] specializes in syntax checks for the Turtle RDF format[1] but confines users to a single format and a specific repository service, limiting real-time collaboration. WebProtégé [13] enables collaboration through a server-based approach but mandates hosting on its servers, necessitating user accounts for collaboration, and it does not support text editing. VocBench [12] supports collaborative SKOS thesaurus editing but involves a complex deployment setup.

Taking from platforms such as Overleaf[2], OntoEditor stores content in a database, generating a unique shareable link. This link fosters collaboration by inviting contributors to work within the same document, offering users the flexibility to choose between individual or collaborative work based on their preferences.

OntoEditor fulfills the ongoing demand for a versatile tool enabling ontology development in any RDF format, syntax parsing, real-time collaboration, and seamless integration with VCS.

[1] `https://www.w3.org/TR/turtle`
[2] `https://www.overleaf.com/learn`

4 OntoEditor: a Collaborative Ontology Editor

OntoEditor is an Online Collaborative Ontology Editor, built on Distributed VCS. It aims to support collaborative ontology development across different RDF serialization formats: Turtle, JSON-LD[3], and RDF/XML. The following discusses its processing workflow as well as the integrated components for empowering collaborative editing.

4.1 OntoEditor Workflow

Figure 2 provides an overarching view of the fundamental steps within OntoEditor. The process commences with user authentication in Git, where credentials are provided. Upon successful authentication, users gain access to remote repository data, including repository names, branches, and file details. Subsequently, upon file and RDF serialization format selection, the editing phase commences.

The unique project link allows new users to view the names and cursor positions of those already connected to the document. This collaborative environment enables multiple users to concurrently edit RDF content, with every change being visible to all collaborators. Additionally, users have the option to individually enable or disable the syntax checker. When activated, the system parses the RDF document, identifying and displaying any syntax errors present. Upon completion of changes, users can commit and push their updates and changes to the distributed repository.

4.2 Collaborative Editing Components

OntoEditor encourages collaborative ontology development among multiple users, driven by essential components that facilitate seamless interaction:
Customizable Editor: CodeMirror[1] as a JavaScript-based Editor was selected for its robust programmable API and advanced editing capabilities such as auto-indentation, auto-completion, syntax highlighting, and search functionalities. As an open-source editor widely used in various projects, CodeMirror inherently supports syntax highlighting for over a hundred programming languages, including Turtle, XML, and JSON-LD. Crucially, it enables collaboration by detecting changes through *onChange* events.
Real-Time Communication Channel: Real-time communication is vital for collaborative editing. WebSocket [5] technology enables immediate and bidirectional data exchange between web browsers (clients) and servers, facilitating seamless interactions. Upon initiating document editing, a WebSocket connection is established. This channel relays all modifications to the server, managing live chat, user details, cursor positions, and notifications among connected users.
ShareDB: To enable seamless collaboration, a real-time database was imperative. After careful research, ShareDB[2] emerged as the optimal choice. ShareDB,

[3] `https://www.w3.org/TR/json-ld11`
[1] `https://codemirror.net/`
[2] `https://share.github.io/sharedb/`

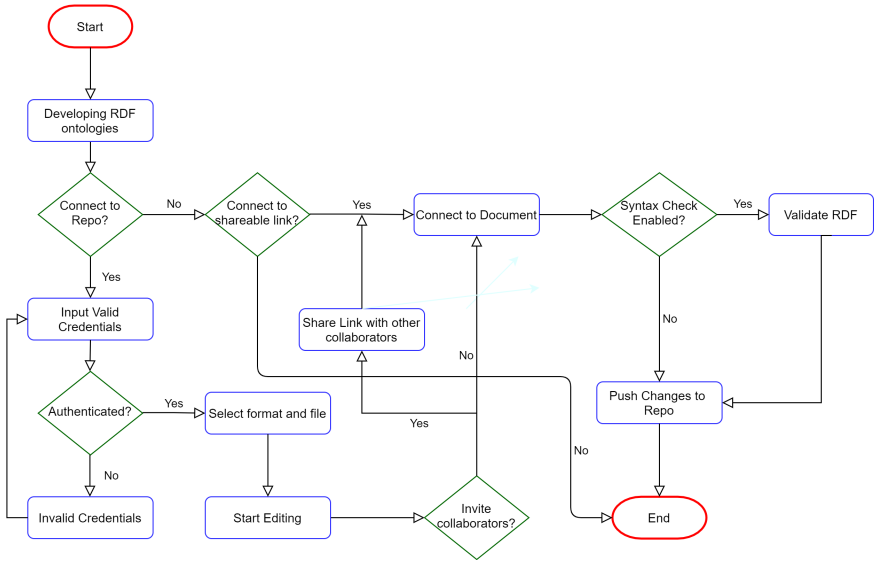

Fig. 2: **OntoEditor's Workflow.** The diagram illustrates the authentication process in Git, file selection, and the initiation of RDF editing. It highlights the collaborative nature facilitated by a unique shareable link, enabling simultaneous editing and syntax checking. Completed changes can be committed and pushed to the remote repository.

built on Operational Transformation (OT) [4], operates as a real-time in-memory database. It stores JavaScript objects on the server and facilitates their sharing among multiple clients through WebSockets. Documents in ShareDB include properties such as Version (incrementing from 0), Type (e.g., OT-text, OT-json1), and Data which is the intended content for storage within the database.

Algorithm 1 was designed to operationalize this approach. Upon a user's initiation of RDF document editing, a new document with an initial version of 0 and the RDF data to be inserted is created in ShareDB. If the document already exists in ShareDB, its existing path is returned to the user. For Operational Transformation, we leverage Plain text Operational Transformation[1]. This Operational Transformation type is utilized for editing plain text documents and supports operations including skipping forward N characters, inserting *str* at the current position, and deleting N characters at the current position.

Clients subscribe to ShareDB documents, updating the document's state with insertions and deletions by modifying the index position and content. Each change increments the version number and is stored in ShareDB. These operations are transmitted over WebSockets, updating local document states.

RDF Validator: OntoEditor validates RDF serialization formats—Turtle, RDF/XML, and JSON-LD—in real-time, allowing immediate syntax error detection while typing. Users can toggle validation on or off as needed, with the tool pro-

[1] `https://github.com/ottypes/text`

Algorithm 1: The pseudo-code of Collaboration

```
Data: inputFile, repoDetails
Result: docChanges,users
users = [ ]
if editingMode then
    path = startEditing(inputFile,repoDetails)
    users.push(name, cursorPosition)
    doc = webSocket(path)
    if doc.subscribe then
        Initialize CodeMirror(doc.value)
        if doc.change then
            user.updatePosition(userCursor)
            if addedText then
                sharedb.submitOp([position, addedText])
            else
                deletedText
            sharedb.submitOp([position, length.deletedText])
            doc.version += 1
        if sharedb.receivedOp then
            if op == sender then
                return
            else
                if op == insertion then
                    codeMirror.replaceRange(newData,position)
                if op == deletion then
                    codeMirror.removeRange(' ', startPosition, endPosition )
    Function startEditing(inputFile, repoDetails)
        if File exists in shareDB then
            shareLink = (repoDetails) + sharedb.fetchDoc()
            return shareLink
        else
            shareLink = (repoDetails) + sharedb.createDoc(inputFile)
            return shareLink
    end
```

viding coherent error messages. Even with syntax errors, users retain the ability to push file changes to the remote repository.

5 Implementation

Considering Figure 3, OntoEditor comprises three key modules, each serving a distinct role within the system. The initial module manages communication with remote repositories, while the second module is designed to enable real-time collaboration among users. Finally, the third module is dedicated to comprehensive syntax validation.

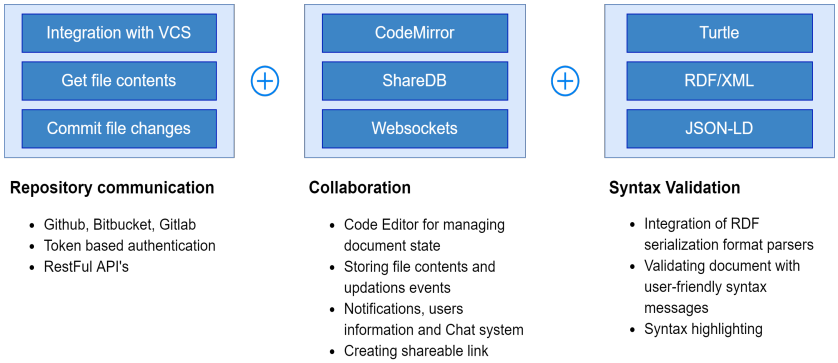

Fig. 3: **OntoEditor's Modules.** OntoEditor comprises three modules: Repository Communication, Collaboration, and Syntax Validation.

5.1 Repository Communication

We utilize GitHub, GitLab, and Bitbucket's APIs for web-based RDF editing, bypassing the need for local Git installation. Authentication requires a username and access token for GitHub and GitLab, while Bitbucket needs an access token with an empty username. Users access repositories and branches and filter files by formats such as *ttl*, *rdfxml*, etc. Available actions include file operations, commits (requiring authentication), and link sharing.

To manage conflicts or concurrent edits, we monitor file history for new commits every 60 seconds. Conflict resolution leverages the Mergely JavaScript library[1], offering users a side-by-side comparison, shown in Figure 4.

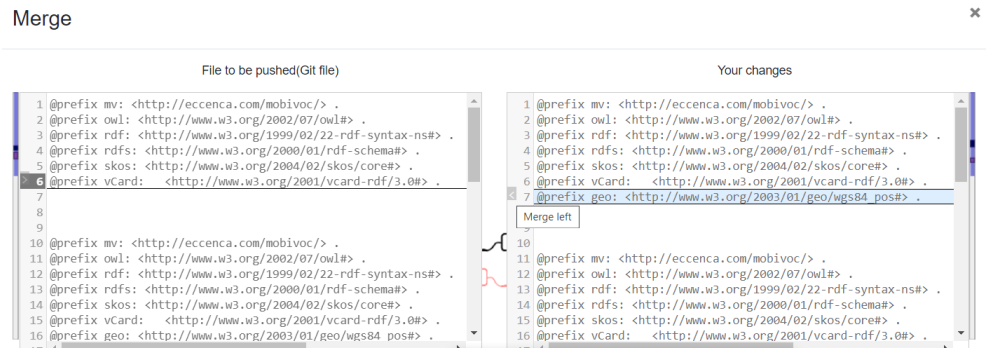

Fig. 4: **Git Conflict Resolution Merge View.** A user interface showcases the user's current document version on the left and the changes retrieved from the hosted Git repository on the right. It facilitates Git conflict resolution by displaying the most recent version of the document if new changes exist.

[1] `https://github.com/wickedest/Mergely`

5.2 Collaboration

This module utilizes Socket.io[2] over WebSockets for real-time editing, cursor tracking, and multi-user communication. Socket.io enables bi-directional, event-driven client-server communication via JavaScript libraries, aimed at simplifying the complexity of editing operations into independent microservices.

Collaboration Service and File Storage in shareDB: To enable real-time collaboration, the file content is stored in our Database. Users initiating edits on the client-side trigger a REST API call to the server, providing Git authentication parameters, file details, and the chosen RDF serialization format. ShareDB is initialized on the server, allowing connections via a separate port. We uniquely identify each file using SHA-1 hashes [3], mirroring how Git stores file information. The hash becomes the document's ID in shareDB. If the document doesn't exist, we create it with an initial version of 0, storing both current data and individual operations.

A unique URL path is generated for each file, containing vital information such as ProjectID, repository details, branch, file name, and RDF serialization format. This link is sent to the client for editing. If the document exists, we send its path to the user, enabling collaboration by sharing the link.

The data is stored in MongoDB for persistence. Locally, ShareDB's in-memory database suffices. Real-time communication begins when a user starts editing, establishing a WebSocket session to the server. Users' names, cursor positions, and document details are maintained on the server and broadcasted to all connected clients. An integrated chat widget allows communication within the system.

Document Updates and Conflict Resolution: ShareDB, as a real-time database, ensures users view the latest document state. Operational Transformation resolves conflicts, managing concurrent editing. CodeMirror's API triggers *onChange* events for any insertion or deletion in the editor.

Insertion operations, demanding an index position and added text, elevate the document's version, while deletions require the index and deletion amount, similarly increasing the version. These actions, applied in ShareDB, are broadcasted to all clients and locally updated. Insert operations synchronize by replacing text using CodeMirror's *replaceRange* function, while deletions are executed by replacing text with an empty string. This process guarantees consistent real-time collaboration across multiple users.

5.3 RDF Validation and Error Notification

OntoEditor utilizes JavaScript parser libraries for real-time validation of various RDF serialization formats (Turtle, RDF/XML, JSON-LD). Users receive instantaneous error messages and can rectify syntax errors seamlessly during collaborative editing of RDF data.

[2] `https://socket.io`

To validate RDF, established parsers are employed: N3.js[1] for Turtle, RDF/XML streaming parser[2] for RDF/XML, and JSON-LD streaming parser[3] for JSON-LD. These parsers operate streamingly, ensuring efficient handling of large documents with limited memory.

During the editing process, users can select their desired format. The chosen parser is activated accordingly, integrated with an *onChange* function to check syntax while typing automatically. The syntax checker can be toggled on or off, with default activation. The syntax checker identifies the format from the URL path and calls the corresponding parser. Parsing occurs in a streaming manner, providing parsed triples and highlighting any syntax errors. Meaningful error messages are displayed atop the editor for immediate user visibility. Upon error correction, a *Syntax correct, all triples parsed successfully* message is shown.

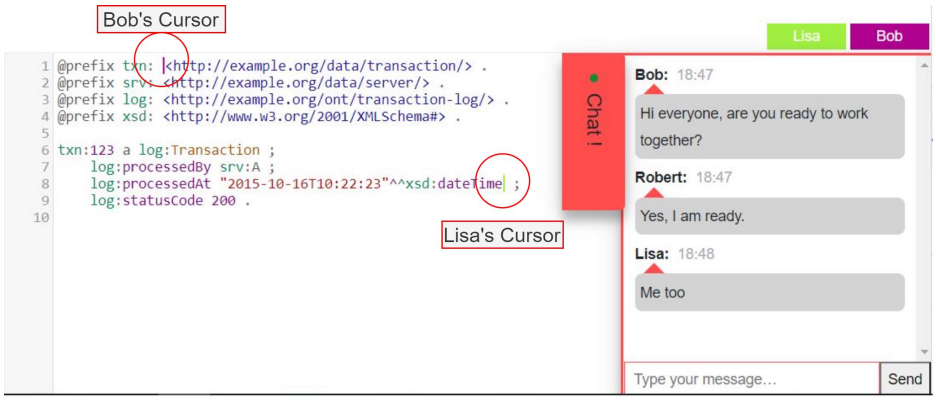

Fig. 5: **Real-Time Collaboration Snapshot.** Robert, Lisa, and Bob are concurrently editing, each represented by their cursor positions. Additionally, they are engaging in communication through an integrated chat widget.

6 Evaluation

OntoEditor underwent comprehensive evaluation through both functional testing and a user study. The functional tests involved experimenting with collaborative editing via an automated typing script on RDF Turtle documents. In contrast, the user study specifically targeted participants from a computer science background to assess their experiences and feedback.

6.1 Functional Testing

This task assesses collaborative editing performance by multiple automated users on RDF Turtle documents. It tests real-time collaboration with consistency using various browsers and clients.

[1] `https://github.com/rdfjs/N3.js`
[2] `https://github.com/rdfjs/rdfxml-streaming-parser.js`
[3] `https://github.com/rubensworks/jsonld-streaming-parser.js`

Objective and Experiment Configuration: The testing aimed to assess real-time collaborative editing under simultaneous input from multiple users subscribed to the same document. Conducted on a Windows 10 machine with a 3rd Gen Intel Core i7-3630 CPU, 2.40 GHz, and 8 GB RAM, the web application was tested across various browsers: Chrome, Firefox, and Opera.
Procedure: Five clients, denoted as tabs A to E, were concurrently opened and tasked with typing different sections of an RDF Turtle file using automated scripts. Initially, an empty document shareable link was generated and accessed by these five clients across separate browser tabs (one client per tab). To assess cross-browser functionality, three clients were opened in Chrome across three tabs, one in Firefox, and one in Opera.

Each client was associated with a distinct segment of the RDF turtle file, enabling simultaneous collaborative editing. To automate typing, a code snippet tailored for each client was utilized. This script, available in our GitHub repository[1], assigned separate instances of the ontology to each client from the shared RDF file. The script prompts for *RDF* input and *Starting time* for execution. Employing an interval function, the script simulates the typing process at variable speeds until completion of the assigned code segment
Results and Discussion: During the experiment, all clients worked concurrently without conflicts, ensuring a consistent document state and robust cross-browser compatibility. Functional tests validated key functionalities, including connection to a unique shareable project link, display of connected users, indication of cursor positions, simultaneous typing by multiple clients, maintenance of a conflict-free document state, and verification of cross-browser compatibility. Client identities from A to E were assigned, and in Figure 6, client A's browser view exhibits connected clients' names and their cursor positions. The conducted typing script affirmed successful and seamless collaboration, enabling consistent and conflict-free document production among all connected clients.

6.2 User Study

The user study presents a comparative analysis of the user experiences between OntoEditor, TurtleEditor, and WebProtégé. Additionally, it comprehensively outlines the evaluation process steps. To gauge the accessibility of our tool, we employed the Concurrent Think-aloud method. This method involved observing participants closely as they performed tasks, allowing us to capture their real-time thoughts and insights.
Participants & Procedure: Nine participants, varying in expertise from basic to advanced in computer science ontology and modeling, took part in the evaluation. They possessed some familiarity with VCS, particularly Git. The evaluation included comprehensive introductions to OntoEditor, TurtleEditor, and WebProtégé for a comparative analysis. Tasks were assigned across all three tools, with continuous monitoring of participants' approaches to solve each task. Participants could access guidance in the *Help* section as needed. After completing tasks, the focus shifted to evaluating user comments on tool usability rather

[1] `https://w3id.org/ontoeditor`

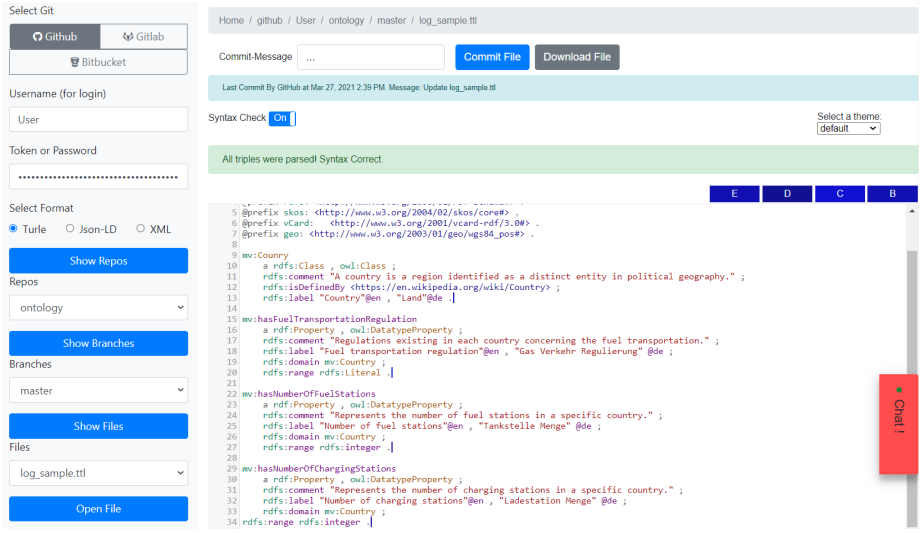

Fig. 6: **Snapshot of Functional Testing Result**: Client A's browser interface displays real-time status after executing an automated typing script. Visible are the names of the four connected users, along with their respective cursor positions, synchronized within the editor.

than just result accuracy. Participants offered suggestions for tool improvement, contributing to future iterations. A survey featuring scaled questions (from 1 to 5) was used to gather detailed user feedback.

Tasks & Questionnaire: The evaluation encompassed nine participants with varying levels of expertise in computer science and Git. They received comprehensive introductions to OntoEditor, TurtleEditor, and WebProtégé for comparative analysis, followed by tasks assigned across all three tools. Continuous monitoring tracked their approaches, with *Help* section guidance available.

Tasks were designed to cover activities from token-based authentication to the final Git commit, executed by three groups. Participants freely chose their preferred GIT VCS platform (GitHub, Bitbucket, or GitLab), completing similar tasks with minor variations in defining properties and instances within different base examples. These base examples in Figure 7 were extended using TurtleEditor and WebProtégé for comparison.

Post-tasks, participants completed an electronic questionnaire, including the USE Questionnaire[1] utilizing a Likert scale. It assessed usefulness, ease of use, ease of learning, and satisfaction. The second section explored specific areas, gathering insights into individual service importance within OntoEditor. Open-response questions captured participant perceptions of pros and cons, influencing future service integration possibilities. The detailed survey questionnaire is available in our project's repository [6].

[1] `https://garyperlman.com/quest/quest.cgi?form=USE`

Group 1	Group 2	Group 3
:Person rdf:type owl:Class ; rdfs:comment "Human"@en; rdfs:label "Person"@en . :Actor rdf:type owl:Class ; rdfs:comment "Acting"@en ; rdfs:label "Actor"@en ; rdfs:subClassOf :Person .	:Media rdf:type owl:Class ; rdfs:comment "Media"@en; rdfs:label "Media"@en . :Movie rdf:type owl: Class ; rdfs:comment "Movie"@en ; rdfs:label "Movie"@en ; rdfs:subClassOf :Media .	:Vehicle rdf:type owl:Class ; rdfs:comment "Transportation Mean"@en ; rdfs:label "Vehicle"@en . :Bus rdf:type owl:Class ; rdfs:comment "Bus"@en ; rdfs:label "Bus"@en ; rdfs:subClassOf :Vehicle .

Fig. 7: **Participants Group Assignments**. Tasks included defining new properties and instances using those base examples.

Results: Participants efficiently completed tasks within 15 to 20 minutes. Post-study, the USE questionnaire revealed high ratings for OntoEditor: usefulness (4.45), ease of use (3.98), ease of learning (4.29), and satisfaction (4.41), indicating substantial usability favorability. Figure 8 shows participant ratings on OntoEditor, emphasizing high ratings for *Collaboration* and *Syntax Validation*. Key findings highlighted high satisfaction with collaboration for multi-user tasks, positive feedback on syntax checking across RDF formats, and suggestions for integrating a user login system and favoring single sign-on for authentication.

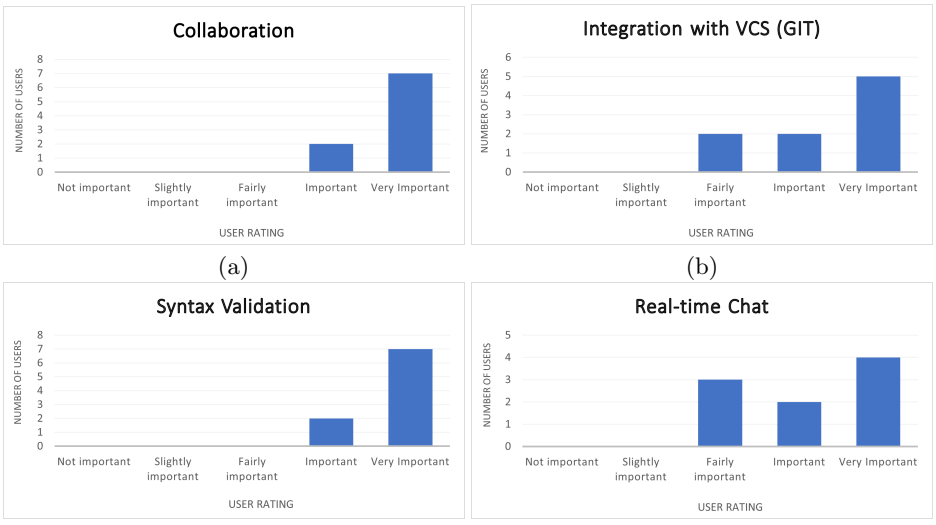

Fig. 8: **User ratings.** User Satisfaction Assessment of OntoEditor.

Comparing OntoEditor, TurtleEditor, and WebProtégé: OntoEditor scored 95%, outperforming both in collaboration and syntax validation. TurtleEditor integrates with VCS but lacks real-time collaboration, while WebProtégé sup-

ports collaboration but lacks synchronization with repositories and textual RDF editing compared to OntoEditor.

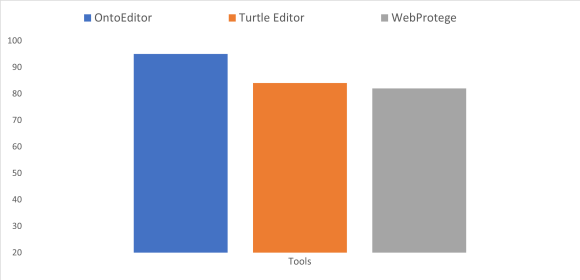

Fig. 9: **Comparative study results**. Showcasing OntoEditor's higher score of 95%, TurtleEditor's 84%, and WebProtégé's 82% in terms of collaboration and syntax validation.

7 Limitations

Throughout project testing tasks and user reviews, OntoEditor has revealed certain limitations, highlighting potential areas for improvement and future tasks:

1. **Lack of user authentication:** OntoEditor lacks its own authentication and user management system for controlling user access and permissions.
2. **No support for ontologies stored locally on file systems:** The primary focus of OntoEditor was to resolve Git conflicts, thus its utilization for ontology development on local file systems was not incorporated.
3. **Inability to edit multiple files via a single project link:** OntoEditor's current incapacity to simultaneously edit multiple files through a single project link hampers collaborative efficiency, especially when parallel edits across multiple files are necessary.
4. **Limited support for serialization formats:** While OntoEditor robustly supports ontology development, its compatibility remains restricted to specific serialization formats, including Turtle, RDF/XML, and JSON-LD.

8 Adoption

An outstanding attribute of OntoEditor is its seamless integration potential within VoCoREG [8], a comprehensive ontology development environment. VoCoREG augments OntoEditor by offering an array of functionalities such as Ontology Metrics, Evolution details, Query services, and Visualization of Ontologies. The integration promises an efficient platform for real-time collaborative editing of various RDF serialization formats among multiple contributors.

9 Conclusion

This paper introduces OntoEditor, a collaborative ontology development tool leveraging Version Control Systems. Its core modules—Repository Communication, Collaboration, and RDF Validation—are integral to its functionality. The Repository Communication module integrates RESTful APIs from GitHub, GitLab, and Bitbucket, ensuring direct user-repository interaction and conflict prevention. The Collaboration module, a separate microservice, enables real-time collaboration via WebSocket, supported by ShareDB for storage. Unique links allow simultaneous editing, coupled with an in-built chat feature for user communication. The RDF Validation module ensures error-free ontology development by real-time syntax validation for RDF serialization formats, enhancing typing accuracy. Empirical evaluations highlighted OntoEditor's standout features: praised collaboration tools, robust syntax validation, and user productivity, especially for Git users. Feedback emphasized improving user integration and implementing single sign-on. Comparative assessments affirmed OntoEditor's superiority over TurtleEditor and WebProtégé in collaboration and syntax validation. OntoEditor emerges as a versatile tool, poised to revolutionize collaborative ontology editing. Its integration potential with VoCoREG expands collaborative development across RDF serialization formats, significantly contributing to ontology development.

10 Future Work

OntoEditor holds potential for advancement in several crucial areas. First, implementing a user authentication system could significantly enhance collaboration by meticulously tracking individual changes. Moreover, integrating a single sign-on authentication method would simplify access to remote repositories, improving user experience and workflow efficiency. Expanding its support to encompass additional RDF serialization formats, such as RDFa and Notation3, stands as another pivotal area for OntoEditor's evolution. Additionally, enabling the platform to import local files for ontology development, independent of VCS, would mark a substantial stride toward enhanced versatility and accessibility.

Acknowledgement

We express gratitude to the Cognitive Internet Technologies Research Center at Fraunhofer for their vital support, as well as to our colleagues and students for their collaborative efforts. Special thanks to ChatGPT [9] for enhancing writing quality, optimizing sentence structure, and eliminating errors in this paper. We also acknowledge its use in summarizing initial notes and proofreading the final draft, extending our appreciation to its developers.

References

1. Berners-Lee, T.: Semantic Web Road Map (September 1998), https://www.w3.org/DesignIssues/Semantic.html, Accessed: 2023-12-01
2. Cyganiak, R., Wood, D., Stones, R.M.L.: Resource Description Framework (RDF): Concepts and Abstract Syntax, https://www.w3.org/TR/rdf11-concepts/
3. Dang, Q.: Secure Hash Standard (SHS), Federal Inf. Process. Stds. (NIST FIPS). National Institute of Standards and Technology, Gaithersburg, MD (2012), https://doi.org/10.6028/NIST.FIPS.180-4
4. Ellis, C.A., Gibbs, S.J.: Concurrency Control in Groupware Systems. SIGMOD Rec. p. 399–407 (1989). https://doi.org/10.1145/66926.66963, https://doi.org/10.1145/66926.66963
5. Fette, I. and Melnikov, A.: The WebSocket Protocol (2011), https://tools.ietf.org/html/rfc6455, Accessed: 2023-11-11
6. Hemid, A.: OntoEditor Survey Form, https://github.com/ahemaid/OntoEditor/blob/main/SURVEY_FORM.pdf, Accessed: 2024-03-21
7. IsSemantic RDF Tools: isSemantic.net: Validate, visualize, generate, and convert structured data, https://issemantic.net/rdf-converter, Accessed: 2023-11-12
8. Khiat, A., Halilaj, L., Hemid, A., Lohmann, S.: VoColReg: A Registry for Supporting Distributed Ontology Development using Version Control Systems. In: 2020 IEEE 14th International Conference on Semantic Computing (ICSC). pp. 393–399 (2020). https://doi.org/10.1109/ICSC.2020.00078
9. OpenAI: ChatGPT 3.5 (2023), https://chat.openai.com/, Large Language Model, Accessed: 2024-03-01
10. Petersen, N., Coskun, G., Lange, C.: TurtleEditor: An ontology-aware web-editor for collaborative ontology development. In: Proceedings of the Tenth IEEE International Conference on Semantic Computing, February 3-5, 2016, Laguna Hills, California, USA (2016), http://dx.doi.org/10.5281/zenodo.35499
11. Prud'hommeaux, E.: RDF Validation Service, http://www.w3.org/RDF/Validator/, Accessed: 2024-01-11
12. Stellato, A., Rajbhandari, S., Turbati, A., Fiorelli, M., Caracciolo, C., Lorenzetti, T., Keizer, J., Pazienza, M.: VocBench: A Web Application for Collaborative Development of Multilingual Thesauri. In: ESWC (2015)
13. Tudorache, T., Vendetti, J., Noy, N.: Web-Protege: A Lightweight OWL Ontology Editor for the Web. In: OWLED (2008)