Universal Library of Engineering Technology **Research Article**

ISSN: 3064-996X | Volume 2, Issue 4

Open Access | PP: 87-92

DOI: https://doi.org/10.70315/uloap.ulete.2025.0204015

Methods and Tools for Developing Collaborative Applications Based on CRDT Types with User Conflict Resolution

Kostadin Almishev

Abstract

Against the background of the rapid expansion of the segment of collaborative software solutions, fueled by the transition to hybrid employment formats and the general digital transformation, the problem of ensuring data consistency in distributed systems is coming to the fore as a fundamental scientific and engineering task. Classical strict consistency schemes are in conflict with the requirements of high availability and fault tolerance, which directly follows from the limitations of the CAP theorem. Conflict-free replicated data types (CRDTs) represent a constructive response to this challenge by providing strict final consistency (SEC) without centralized coordination. The purpose of the research is to systematize and analyze approaches and tools for building collaborative applications based on CRDT, as well as to formulate a multi-level model for resolving semantic conflicts that arise at the level of user intentions. Methodologically, the work is based on a systematic review of academic publications, a comparative study of the leading CRDT libraries (Yjs, Automerge) and a content analysis of industry reports. A comprehensive analysis of the theoretical foundations of CRDT, their practical incarnations, and existing conflict resolution strategies has shown the insufficiency of automatic mechanisms for handling semantic collisions. In response, a three—level Data—Logic-Representation architecture is proposed, which distinguishes between automatic merging at the data level, logical identification of semantic contradictions and their resolution at the user interface layer through adapted UI/UX patterns. The findings indicate that this principle of organization increases the predictability of the system's behavior and makes collaboration more intuitive. The research materials are addressed to other researchers, architects of distributed systems and software developers.

Keywords: *CRDT, Collaborative Applications, Distributed Systems, Strict Final Consistency, Conflict Resolution, Semantic Conflicts, Yjs, Automerge, User Interface, Local-First.*

INTRODUCTION

The contemporary digital economy demonstrates an unprecedented intensification of technological reliance on tools for collaborative activity. The global segment of collaboration software maintains a stable upward trajectory: as of 2024, its size is estimated at USD 36.1 billion [1]. The driving forces are structural transformations of organizational labor models—the large-scale institutionalization of hybrid and remote forms of employment [4]—combined with the acceleration of digital transformation programs across industries [5, 6]. Specialized analytical reviews by leading consulting companies confirm the long-term nature of these trends and the formation of sustained demand for solutions that ensure continuous, low-latency interaction of distributed teams in real time [2, 3].

The growing need for collaborative platforms exacerbates a fundamental engineering dilemma formalized by the CAP theorem (Consistency, Availability, Partition Tolerance): in a distributed system, it is impossible to simultaneously guarantee strict data consistency, full availability, and tolerance to network partitions [7, 8, 10]. Classical architectures based on centralized DBMSs and locking to preserve strict consistency inevitably reduce availability under network failures—a trade-off incompatible with expectations for modern applications, where offline-first scenarios and instantaneous interface responsiveness are required [9, 11]. In contrast, the paradigm of eventual consistency is applied, aiming for high availability at the cost of temporary divergence of replica states; however, early incarnations of this approach were often ad hoc and error-prone, generating unpredictable, hard-to-diagnose conflicts and data states [10].

The resolution of this problem was provided by conflict-free replicated data types (CRDT). CRDT are defined as data structures that support replication across multiple nodes and allow independent, concurrent updates without coordination between replicas [12, 14]. Their algebraic invariants ensure asymptotic convergence of the states of all copies. This result is formally described by the model of Strong Eventual Consistency (SEC): any two replicas that have processed the same set of updates—even in different orders—arrive at an identical state [10, 13].

However, the practical adoption of CRDT in industrial solutions has revealed a substantial scientific and applied gap. Although CRDT effectively eliminates conflicts at the level of merging the data structures themselves (e.g., when elements are added to a set concurrently), they do not address the problem of semantic collisions arising from contradictory user intentions [17]. A characteristic situation is as follows: one participant edits the text of a paragraph while another concurrently deletes it. From the CRDT perspective, the merge is correct (for example, under a deletion-wins strategy), but from the user perspective this results in the loss of the product of their labor—a semantically undesirable outcome. The existing literature presents the mathematical foundations and algorithmic constructions of CRDT in detail, but does not offer a systematic model for identifying, representing, and resolving such user conflicts at the application level.

The purpose of the study is to systematize and provide an analytical review of methods and tools for building collaborative applications based on CRDT and to present a multilevel model for resolving semantic conflicts manifested at the level of user intentions.

The author's hypothesis is that the application of a multilevel approach—with separation of tasks into the data layer (automatic merging via CRDT), the application logic layer (detection of semantic collisions within the application), and the presentation layer (UI/UX patterns that involve the user in resolution)—makes it possible to create predictable, robust, and intuitive systems for collaborative work.

The scientific novelty consists in proposing a comprehensive model for resolving user conflicts in CRDT systems, integrating theoretical principles of distributed systems, analysis of contemporary development tooling, and principles of human–computer interaction design.

MATERIALS AND METHODS

The study has an integral, interdisciplinary nature and relies on a heterogeneous yet coherent methodology that combines several analytical frameworks for a comprehensive examination of the stated problem area.

As a theoretical foundation, a systematic literature review was conducted. Peer-reviewed publications, materials from leading computer science conferences (IEEE, ACM), and Springer proceedings over the years were analyzed. Priority was given to foundational works that establish the formal properties of CRDT and the model of Strong Eventual Consistency. In addition, recent studies on the formal verification of CRDT algorithms and assessment of their applicability in complex systems were taken into account, which made it possible to delineate the current state and boundaries of theoretical knowledge in this area.

To study the practical dimension of the topic—development tooling—a comparative analysis was applied. Key libraries that are de facto standards for building CRDT-based collaborative applications, in particular Yjs and Automerge, were examined. The analysis covered their architecture, data models, stated synchronization mechanisms, and API interfaces based on official technical documentation. Performance was evaluated using the results of independent benchmarks and comparative studies that consider operation processing time, memory consumption, and the volume of transmitted data.

To identify the macroeconomic and social context, content analysis was employed. Analytical reports and technology forecasts from leading consulting agencies, including Gartner, McKinsey, and Deloitte, were reviewed. This stage made it possible to capture the key trends—growth of remote employment and acceleration of digital transformation—which directly shape the requirements for modern collaborative software and confirm the relevance of the study.

RESULTS AND DISCUSSION

The foundation of fault-tolerant collaborative systems is formed by two classes of CRDT that differ in their replication strategy: state-based (state-oriented) and operation-based (operation-oriented).

State-based CRDT, or Convergent Replicated Data Types, achieve consistency through periodic transmission between replicas of the full state of an object or its delta, after which the local copy performs a merge. The formal guarantee of convergence is provided by the requirement that the set of reachable states forms a joint semi-lattice. From this follows the existence of a least upper bound for any two states S1 and S2, and the merge function computes the LUB (Least Upper Bound). For correctness, it must be associative, commutative, and idempotent [10, 15]. Due to these properties, CvRDT is resilient to losses, duplicates, and arbitrary message reordering: the system still converges to a single consistent state.

Operation-based CRDT, or Commutative Replicated Data Types (CmRDT), take the opposite path: instead of transmitting states, replicas disseminate the operations (mutations) that change local data. Convergence under this approach requires stricter assumptions. First, the communication environment must ensure delivery of each operation to all replicas, typically while preserving causal order and without duplication. Second, all concurrent operations (not related by a happened-before relationship) must commute [10, 16].

The choice between these approaches constitutes a significant architectural trade-off; the main differences and the implications of such decisions are summarized in Table 1.

Table 1. Comparative analysis of the CvRDT and CmRDT approaches (compiled by the author based on [10, 15, 17, 20, 23]).

Criterion	**CvRDT (State-based)**	**CmRDT (Operation-based)**
Synchronization mechanism	Transfer of the full or delta state of the object.	Transfer of individual operations (mutations).
Network requirements	Minimal. Tolerant to message duplication, loss, and reordering.	High. Requires reliable message delivery (no duplicates, with causal ordering).
Message size	Potentially large (entire state), but can be optimized using delta states (δ-CRDTs).	Small (a single operation), but may include metadata to ensure ordering.
Implementation complexity	Relatively low. Merge logic is encapsulated in the merge function.	High. Requires implementation of a complex and reliable communication subsystem.
Type examples	G-Counter (growing counter), PN-Counter (counter with increment/decrement), G-Set (growing set), LWW-Register (last-write-wins register).	Op-based Counter, Op-based Observed-Remove Set (set with removal).

The theoretical foundations of CRDT have received a mature engineering implementation in a number of libraries; among them, Yjs and Automerge have de facto established an industry standard due to a combination of maturity, performance, and broad ecosystem support.

Yjs is a high-performance CRDT implementation specifically optimized for strict real-time scenarios. Architecturally, it embodies an advanced operation-based model in which operations are aggressively compacted, reducing network and computational overhead. Its core is the Y.Doc object, a container for a collection of shared types, including Y.Map, Y.Array, and Y.Text. Synchronization between clients is implemented via interchangeable transport providers (e.g., WebSocket or WebRTC), which makes the stack network-agnostic [25]. An important feature is the Awareness protocol, a separate channel for ephemeral information (cursor positions, user statuses, etc.) that is not part of the document and does not require persistence in the history [20]. Public benchmarks demonstrate a substantial advantage of Yjs in operation processing speed and memory efficiency, which makes it preferable for high-load editors [21, 25].

Automerge follows a different paradigm. A document is represented as a JSON-like structure, and the library, unlike Yjs, preserves a complete, immutable history of all modifications—in the spirit of Git. This design naturally supports viewing the evolution of a document, branching, and subsequent merging of versions. The architecture relies on a high-performance Rust core (with portability, including WebAssembly) and a Repo object that abstracts storage management (storage adapters) and network interaction (network adapters). The emphasis on immutable states and a local-first approach makes Automerge a compelling choice for systems in which auditability, strict versioning, and reliable offline operation are critical [22].

The choice between the libraries is strategic and predetermines the application architecture. Yjs is oriented toward maximizing the speed of converging the current state, which is an optimal option for highly dynamic interfaces. Automerge essentially provides capability for tasks with strict versioning requirements, but at the cost of increased overhead. These architectural differences and the corresponding performance trade-offs are illustrated in Fig. 1.

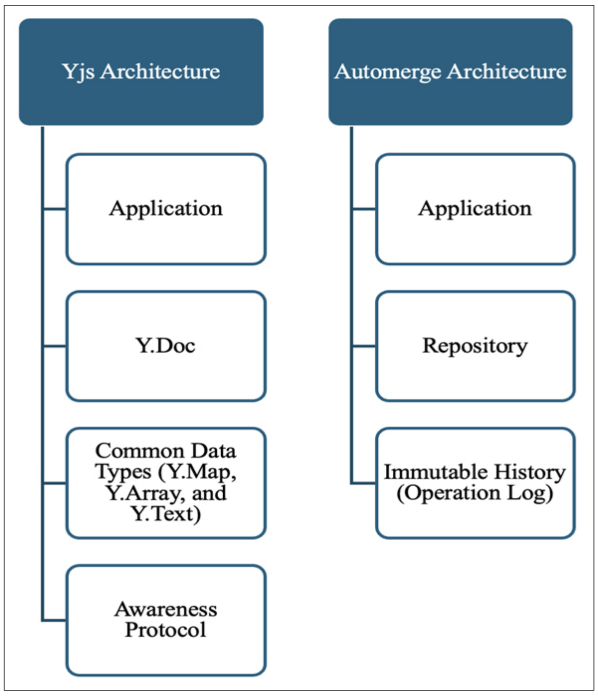

Fig. 1. Architectural comparison diagram of Yjs and Automerge (compiled by the author based on [21, 22, 25, 28, 29]).

As can be seen from Fig. 1, Yjs uses a central Y.Doc with pluggable providers, whereas Automerge uses a Repo to orchestrate documents (DocHandles), each of which contains an immutable history of changes.

The mathematical properties of CRDT guarantee the syntactic convergence of replicas; however, the interpretation of user actions and domain semantics remain outside their scope. This leads to a fundamental distinction between two classes of conflicts. First, data-level conflicts: they are eliminated automatically due to the algebraic invariants of CRDT. For example, when different users concurrently add different elements to a shared set, the resulting state correctly includes both elements. Second, semantic conflicts: a formally consistent union of operations can lead to a state that contradicts the application logic or the intentions of some participants. Because CRDT do not take into account business rules and the execution context of operations, such situations remain outside their model [17, 23, 24].

To partially mitigate semantic effects at the data level, several standard—but limited in applicability—strategies are used. The Last-Writer-Wins (LWW) register tags each write with a timestamp and, under concurrency, selects the value with the maximal timestamp; determinism is achieved at the cost of silent information loss, when the result of one user's work is irreversibly overwritten by the result of another [14]. In a multi-value (MV) register, all concurrent writes are preserved, preventing data loss, but shifting subsequent conflict resolution to the client application—via user choice or embedded merge logic [14]. For sets, Add-Wins / Remove-Wins rules are often used: in the Add-Wins variant, if the addition of an element e and its removal occur concurrently, the add operation takes precedence and e remains in the set. Such a policy is natural for certain scenarios (e.g., a shared shopping cart), but in other contexts it can lead to the resurrection of elements that have already been deleted.

A consolidated comparison of these basic strategies is provided in Table 2.

Table 2. Classification of strategies for automatic resolution of semantic conflicts (compiled by the author based on [11]).

Strategy	Description	Advantages	Disadvantages
Last-Writer-Wins (LWW)	The operation with the highest timestamp is applied.	Determinism, ease of implementation.	Data loss (one user's work is overwritten).
Multi-Value (MV)	All conflicting values are retained for subsequent resolution.	Guaranteed preservation of all data.	Requires explicit resolution on the client side, complicates logic.
Add-Wins Set	Under concurrent add(e) and rmv(e) operations, add(e) takes precedence.	Intuitive behavior for certain scenarios (e.g., shopping cart).	May lead to unintended restoration of deleted data.

The limitations of existing strategies necessitate a reconsideration of the initial premises and a transition from a paradigm of conflict-freeness to a paradigm of meaningful conflict handling. At its core lies the principle of capturing user intent. A conflict is interpreted not as a failure, but as an informative signal that multiple participants have performed incompatible actions [19]. Instead of automatically eliminating one of the versions, the system should retain all conflicting operations and delegate the choice of a substantive resolution either to the application or to the user themself [18].

To impart methodological rigor to this approach, a multilevel model is proposed that structures the process of conflict handling in a collaborative environment. The model distributes functions across the system levels—from basic data synchronization to the interface level of user interaction—thereby establishing clear boundaries of responsibility and a coherent conflict-resolution scenario (see Fig. 2).

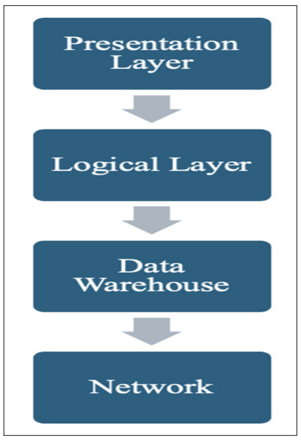

Fig. 2. Diagram of the proposed multi-level model of conflict resolution (compiled by the author based on [18, 19]).

The architectural model of the system is organized as three interconnected layers—data, logic, and presentation. Their coordinated functioning simultaneously ensures the formal consistency of replicas and human-oriented resolution of semantic contradictions that inevitably arise during collaborative editing.

At the data layer, CRDT types operate, in particular those from the Yjs and Automerge libraries. Its only constraint is a mathematically rigorous, deterministic aggregation of states or operations between replicas that guarantees syntactic consistency across all copies. This layer is fundamentally separated from the problem domain: it does not encapsulate business rules and, consequently, does not operate with the semantics of the changes being made.

The logic layer is implemented in application code as an observer of the stream of changes arriving from the data layer. It interprets each operation in the context of the current state and specified business constraints, identifying semantic rather than syntactic collisions. For example, when a text edit of a related task is received, the logic checks whether the task has been deleted by that time. Upon detecting a divergence of intentions, the logic layer neither annuls operations nor performs automatic reconciliation; instead, it forms a semantic conflict event, accumulating metadata about the conflicting actions and their causal–temporal structure. Such a decomposition shifts the choice of the correct interpretation from the algorithmic plane to a controlled user decision-making procedure.

The presentation layer is responsible for user interaction and for materializing conflict events into understandable interface objects. Upon receiving a signal about a semantic conflict, the interface, first, delicately notifies the user—for example, by contextually highlighting the relevant elements;

second, makes the nature of the divergence visually clear; and third, provides tools for deliberate resolution. The practical implementation relies on established UI/UX patterns from version control systems, primarily Git [14, 18]. Key methods include parallel presentation of two alternative versions with highlighted differences (side-by-side diff), three-way merging with simultaneous display of the base, local, and remote versions and the aggregated result in a central area (three-way merge), as well as explicit choice operations—accept my changes, accept remote changes, merge changes. In addition, interactive editing of the final variant is permissible, enabling the combination of fragments from both versions and thereby reducing the loss of information and context [20].

Consequently, the effectiveness of collaborative work is determined not by the degree of smartness of automatic heuristics, but by the quality of interaction design: how quickly, transparently, and with minimal cognitive cost the user understands the nature of the conflict and makes a decision. The proposed stratification—CRDT-backed consistency at the data level, detection of semantic contradictions at the logic level, and human-centered resolution at the presentation level—institutionalizes the separation of responsibility between machine correctness and user judgment, increasing the reliability and predictability of collective editing [17, 19].

In a UI scenario for task management, a concurrent change arises: User A renames a task, while User B synchronously marks it as completed. Instead of forcing auto-resolution, the system initiates a dialog offering an informed choice: (1) Accept the new name and the status completed, (2) Keep the previous name with the status completed, (3) Accept the new name but keep the task incomplete.

CONCLUSION

The conducted study demonstrates that conflict-free replicated data types (CRDT) constitute a reliable, strictly formalized foundation for creating modern collaborative systems with high availability and robust operation in offline scenarios. Comparing theoretical principles with practical implementations—using the examples of the Yjs and Automerge libraries—has revealed the technological maturity of the approach and the presence of a well-developed ecosystem of tools for developers.

At the same time, the principal outcome of the work consists in clarifying the limits of applicability of CRDT guarantees: while ensuring convergence at the level of syntactic merging of replicas, they do not eliminate semantic contradictions generated by competing user intentions. Typical strategies such as Last-Writer-Wins lead to the loss of meaningful data, whereas Multi-Value merely shifts the burden of resolving ambiguity to the application level.

Therefore, the stated objective has been achieved: methods and means of CRDT-based development have been systematized, and a multilevel scheme for handling user conflicts has been proposed. The model, which decomposes the process into three interconnected levels—data (automated merging via CRDT), logic (detection of semantic collisions), and presentation (UI/UX patterns for user participation in resolution)—ensures effective handling of semantic conflicts. This confirms the advanced hypothesis that a structured approach increases the predictability, reliability, and usability of collaborative applications.

The practical significance of the results lies in providing architects and developers with a clear conceptual framework for design: instead of searching for a universal automatic conflict-resolution algorithm, the emphasis shifts toward systems capable of promptly detecting semantic collisions and offering the user intuitive means for their resolution. Promising directions for further research include the formal verification of protocols for resolving user conflicts and the use of machine learning methods for predictive identification of potential semantic collisions based on analysis of behavioral patterns.

REFERENCES

1. Team Collaboration Software Market | Industry Report, 2030 | Grand View Research. Retrieved from: https://www.grandviewresearch.com/industry-analysis/team-collaboration-software-market (date accessed: September 12, 2025).

2. Collaboration Software Market Size, Industry Report 2025–2034 | Global Market Insights. Retrieved from: https://www.gminsights.com/industry-analysis/collaboration-software-market (date accessed: September 15, 2025).

3. Cloud Meeting and Team Collaboration Global Market Forecast Report 2024–2029 with Key Findings from UC and AI Decision-Maker Surveys, Hybrid Work Trends and Investment Priorities | GlobeNewswire. Retrieved from: https://www.globenewswire.com/news-release/2025/06/25/3104840/28124/en/Cloud-Meeting-and-Team-Collaboration-Global-Market-Forecast-Report-2024-2029-with-Key-Findings-from-UC-and-AI-Decision-Maker-Surveys-Hybrid-Work-Trends-and-Investment-Priorities.html (date accessed: September 18, 2025).

4. Development in the Future of Work: 2025 Learning Perspective [PDF] | McKinsey & Company. Retrieved from: https://www.mckinsey.com/~/media/mckinsey/featured%20insights/people%20in%20progress%20blog/learning%20trends%202025/2025_mckinsey%20learning%20perspective.pdf (date accessed: September 22, 2025).

5. McKinsey Global Institute. (2020, November 23). What's next for remote work: An analysis of 2,000 tasks, 800 jobs, and nine countries. Retrieved from: https://www.mckinsey.com/featured-insights/future-of-work/whats-next-for-remote-work-an-analysis-of-2000-tasks-800-jobs-and-nine-countries (date accessed: September 26, 2025).

6. Team Collaboration Software Market Size & Outlook, 2025–2033 | Straits Research. Retrieved from: https://straitsresearch.com/report/team-collaboration-software-market (date accessed: October 1, 2025).
7. Technology Trends Outlook 2024 [PDF] | McKinsey & Company. Retrieved from: https://www.mckinsey.com/~/media/mckinsey/business%20functions/mckinsey%20digital/our%20insights/the%20top%20trends%20in%20tech%202024/mckinsey-technology-trends-outlook-2024.pdf (date accessed: October 7, 2025).
8. McKinsey & Company. (2025, July 22). McKinsey technology trends outlook 2025 (The top trends in tech). Retrieved from: https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/the-top-trends-in-tech (date accessed: October 14, 2025).
9. The Future of Work | Deloitte. Retrieved from: https://www.deloitte.com/global/en/services/consulting/collections/future-of-work.html (date accessed: October 20, 2025).
10. Almeida, P. S. (2024). Approaches to conflict-free replicated data types. ACM Computing Surveys, 57(2), 1–36.https://doi.org/10.1145/3695249.
11. Preguiça, N., Baquero, C., & Shapiro, M. (2018). Conflict-free replicated data types (CRDTs). In S. Sakr & A. Zomaya (Eds.), Encyclopedia of Big Data Technologies (pp. 1–10). Springer.https://doi.org/10.1007/978-3-319-63962-8_185-1.
12. Kaki, G., Prahladan, P., & Lewchenko, N. V. (2022). RunTime-assisted convergence in replicated data types. In Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22) (pp. 364–378). ACM.https://doi.org/10.1145/3519939.3523724.
13. Brauße, F., Collins, P., & Ziegler, M. (2022). Computer science for continuous data: Survey, vision, theory, and practice of a computer analysis system. In F. Boulier, M. England, T. M. Sadykov, & E. V. Vorozhtsov (Eds.), Computer Algebra in Scientific Computing (CASC 2022) (pp. 62–82). Springer.https://doi.org/10.1007/978-3-031-14788-3_5.
14. Mao, Y., Liu, Z., & Jacobsen, H.-A. (2022). Reversible conflict-free replicated data types. In Proceedings of the 23rd ACM/IFIP International Middleware Conference (Middleware '22) (pp. 295–307). ACM.https://doi.org/10.1145/3528535.3565252.
15. Almeida, P. S. (2024). Approaches to conflict-free replicated data types. ACM Computing Surveys, 57(2), 1–36.https://doi.org/10.1145/3695249.
16. Prymushko, A., et al. (2025). Efficient state synchronization in distributed electrical grid systems using conflict-free replicated data types. IoT, 6(1), 6.https://doi.org/10.3390/iot6010006.
17. Weidner, M., et al. (2022). Collabs: A flexible and performant CRDT collaboration framework(arXiv:2212.02618). https://doi.org/10.48550/arXiv.2212.02618.
18. Nieto Rodriguez, A. (2023). Conflict-free Replicated Data Types have Abstract Data Types [PDF]. Retrieved from: https://iris-project.org/pdfs/2023-phd-nieto.pdf (date accessed: November 2, 2025).
19. Rault, P.-A., Ignat, C.-L., & Perrin, O. (2023). Access control based on CRDTs for collaborative distributed applications. In 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) (pp. 1369–1376). IEEE. https://doi.org/10.1109/TrustCom60117.2023.00187.
20. Saquib, N., Krintz, C., & Wolski, R. (2022). Ordering operations for generic replicated data types using version trees. In Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '22) (pp. 39–46). ACM.https://doi.org/10.1145/3517209.3524038.
21. Zhou, K., & Zhang, L. (2025). Step-wise formal verification for LLM-based mathematical problem solving(arXiv:2505.20869).https://doi.org/10.48550/arXiv.2505.20869.
22. Zeller, P., Bieniusa, A., & Poetzsch-Heffter, A. (2014). Formal specification and verification of CRDTs. In Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2014) (pp. 33–48). Springer.https://doi.org/10.1007/978-3-662-43613-4_3.
23. Conflict-free Replicated Data Types have Abstract Data Types [PDF] | IRIS Project. Retrieved from: https://iris-project.org/pdfs/2023-phd-nieto.pdf (date accessed: November 2, 2025).
24. Quick Start (Introduction) | Yjs Docs. Retrieved from: https://beta.yjs.dev/docs/introduction/ (date accessed: November 16, 2025).
25. Shared Types | Yjs Docs. Retrieved from: https://beta.yjs.dev/docs/getting-started/working-with-shared-types/ (date accessed: December 9, 2025).