Specification of team software project

Department of Software Engineering

Faculty of Mathematics and Physics, Charles University

Solvers: Bc. Ondřej Krsička

Study program: Computer Science - Software and Data Engineering

Project title: Using CRDTs to enable collaborative editing in Denicek

Project type: Research project

Supervisor: Mgr. Tomáš Petříček, Ph.D.

Consultant: Jonathan Edwards (Denicek co-author)

Expected start: 1.11.2025

Expected end: 1.7.2026

1 Introduction

This project is part of ongoing research on end-user programming substrates, building on the *Denicek* system presented at ACM UIST 2025 [1]. The research explores how Conflict-free Replicated Data Types (CRDTs) can enable truly local-first collaborative editing for document-oriented programming environments.

Denicek [1] is a computational substrate for document-oriented end-user programming that supports collaborative editing, programming by demonstration, incremental recomputation, and schema change control. However, the current *Denicek* synchronization layer relies on Operational Transformation (OT), which is complex, error-prone, and requires a central synchronization server—violating the principles of local-first software [4].

This project creates *MyDenicek*, an alternative backend built on CRDTs, and *MyWebnicek*, a web-based programming system demonstrating its capabilities. The goal is to provide the same end-user programming experiences as the original *Denicek* while enabling peer-to-peer synchronization without central coordination. Both components are implemented in TypeScript, maintaining clear separation between the CRDT substrate and the user interface to enable future development of additional programming environments.

The project is inspired by *MyWebstrates* [5], a local-first, CRDT-based alternative to Webstrates [6], which demonstrated the viability of CRDT-based approaches for similar collaborative document systems.

1.1 Research context and motivation

End-user programming environments have traditionally struggled with collaboration. While users can create documents, record macros, and build simple applications, sharing these artifacts and collaborating in real-time introduces significant complexity. The original *Denicek* addresses this through Operational Transformation, but OT systems require:

- A central server for transformation ordering

- Complex transformation functions that are difficult to verify
- Careful handling of concurrent operations to avoid anomalies

CRDTs offer an alternative approach where:

- Documents can be modified offline and synchronized later
- No central server is required for consistency
- Merge semantics are mathematically guaranteed to converge

This research investigates whether CRDTs can provide equivalent or better user experiences for collaborative document-oriented programming, while maintaining the full range of *Denicek* features.

1.2 Relation to other projects

This project directly builds on *Denicek* [1] and its web-based implementation *Webnicek*. The system design incorporates lessons from:

- **Loro CRDT** [7]: The underlying CRDT library providing tree-structured data synchronization
- **MyWebstrates** [5]: Demonstrated local-first architecture patterns for collaborative systems
- **Martin Kleppmann's JSON CRDT** [8]: Influenced our decision to use ID-based node addressing

The primary deliverable (*MyDenicek*) will be a reusable TypeScript library that can serve as the foundation for future end-user programming environments. *MyWebnicek* serves as both a demonstration of the library's capabilities and a prototype for evaluating whether CRDT-based synchronization can support the full range of *Denicek* programming experiences.

2 Project description

The core innovation of this project is adapting the *Denicek* document model to work with CRDTs while preserving support for complex editing operations like wrapping, schema changes, and programming by demonstration.

2.1 Document model

The *MyDenicek* document model represents structured documents via a `DocumentView` class that encapsulates the tree structure and provides read-only access through methods:

```
class DocumentView {
  getRootId(): string | null;
  getNode(id: string): NodeData | null;
  getChildIds(parentId: string): string[];
  getParentId(nodeId: string): string | null;
  getAllNodeIds(): string[];
  *walkDepthFirst(): Generator<{ node: NodeData; depth: number; parentId: string | null
     }>;
}

// Node data types (no children array - use getChildIds instead)
interface ElementNodeData {
  id: string;
  kind: "element";
  tag: string;
  attrs: Record<string, unknown>;
}

interface ValueNodeData {
  id: string;
  kind: "value";
  value: string;
}

type NodeData = ElementNodeData | ValueNodeData;
```

Internally, the document is stored as a `LoroTree`—Loro's native movable tree CRDT that handles concurrent structural edits, move operations, and conflict resolution automatically. The `DocumentView` class is the *public API* that:

1. **Hides CRDT internals**: No Loro types are exposed; applications work with plain TypeScript objects.
2. **Enables O(1) lookup**: Internal index maps allow efficient node, parent, and children lookups.
3. **Prevents direct mutation**: Users access data through methods instead of property access.
4. **Simplifies rendering**: React components receive a stable view for efficient diffing.

2.2 Architecture overview

The system follows a layered architecture that hides CRDT implementation details from application code:

```
+---------------------------------------------------------+
|                   MyWebnicek (React App)                 |
|         React 19 + Fluent UI + Custom Components        |
+---------------------------------------------------------+
|                 mydenicek-react-v2                       |
|     DenicekProvider, useDocumentState, useRecording     |
+---------------------------------------------------------+
|                  mydenicek-core-v2                       |
|    DenicekDocument, DenicekStore, DenicekModel          |
+---------------------------------------------------------+
|              Internal: LoroDocWrapper                    |
|           (Loro types hidden from public API)              |
+------------------------------------------------------------+
|                     loro-crdt                               |
|              (Third-party CRDT library)                     |
+------------------------------------------------------------+
```

Key architectural decisions:

- **No Loro types in public API**: The `LoroDocWrapper` class isolates all Loro-specific code, allowing the CRDT implementation to be changed without affecting application code.
- **Snapshot-based reads**: Applications read document state through plain JSON snapshots, not CRDT objects.
- **Callback-based mutations**: All modifications go through `document.change(model => {...})`, ensuring proper transaction handling and undo support.

2.3 Functional requirements

2.3.1 MyDenicek Core Library

FR-01: Document Representation The library shall represent HTML-like structured documents with element nodes (containing tags, attributes, and ordered children) and value nodes (containing text content).

FR-02: Node Operations The library shall support the following primitive operations:

- Add child node (element or value) to a parent
- Delete node
- Update node tag
- Update node attribute
- Edit value content (character-level text editing)
- Wrap node in new parent element
- Add sibling node (before or after reference node)

FR-03: CRDT Synchronization The library shall provide export/import methods for synchronizing document state between peers without central coordination.

FR-04: Undo/Redo The library shall maintain undo history and support undo/redo operations with configurable merge intervals for grouping related changes.

FR-05: Subscriptions The library shall support subscribing to document changes (both local and remote) and to patch-level events for recording.

FR-06: History/Checkout The library shall maintain version history and support checking out previous document states (time travel).

FR-07: Recording/Replay The library shall record document modifications as generalized patches (with variable node IDs like `$0`, `$1`) and replay recorded scripts on different starting nodes—enabling programming by demonstration.

2.3.2 MyWebnicek Application

FR-08: Document Renderer The application shall render the document tree as interactive HTML elements, reflecting the current document state.

FR-09: Navigation Users shall be able to navigate through the document tree, selecting nodes for inspection or modification.

FR-10: Command Interface Users shall be able to perform primitive actions through a command interface (command palette or direct controls).

FR-11: History View The application shall display the history of edit actions, allowing users to understand how the document evolved.

FR-12: Real-time Collaboration Multiple users shall be able to edit the same document simultaneously, with changes synchronized via WebSocket connection to a sync server.

FR-13: Conflict Visualization When conflicts occur during merge, the application shall provide visual indication of resolved conflicts and their outcomes.

FR-14: Recording Interface Users shall be able to start/stop recording their actions and replay recorded scripts on selected nodes.

2.4 Non-functional requirements

NFR-01: Response Time Local document operations shall complete within 50ms. Snapshot generation shall complete within 100ms for documents with up to 10,000 nodes.

NFR-02: Synchronization Latency Changes shall propagate to connected peers within 500ms under normal network conditions.

NFR-03: Offline Support The application shall remain fully functional when offline, queuing changes for later synchronization.

NFR-04: Browser Compatibility The web application shall function correctly on current versions of Chrome, Firefox, Safari, and Edge.

NFR-05: Type Safety All library code shall be written in TypeScript with strict mode. No use of `any` type.

NFR-06: Testability Core library functions shall be unit-testable. UI features shall be testable via Playwright E2E tests.

NFR-07: Documentation All public APIs shall be documented with JSDoc comments.

NFR-08: Reproducibility A developer shall be able to set up the project and run tests within 10 minutes.

Concurrent Operations	Resolution	Rationale
Wrap(A) vs Wrap(B) same node	Single wrapper, LWW on tag	Deterministic wrapper ID
Add Child vs Add Child	Both children added	Random unique IDs
Rename Tag vs Rename Tag	One tag wins (LWW)	Standard CRDT semantics
Edit Value vs Edit Value	One value wins (LWW)	Register semantics
Wrap vs Add Child	Both succeed	Operations are independent
Delete vs Edit	Delete wins	Standard delete semantics

Table 1: Conflict resolution behavior for concurrent operations

2.5 Conflict resolution semantics

A critical aspect of CRDT-based systems is defining how concurrent operations resolve:

The wrap operation's deterministic ID generation (`wrap-${nodeId}`) is a key design decision that prevents "double wrapping" when two users concurrently wrap the same node with different tags.

3 Platform and technologies

3.1 Programming Languages

- **TypeScript** — All code with strict mode enabled
- **HTML/CSS** — Web interface markup and styling

3.2 Frameworks and Libraries

3.2.1 Core Library (mydenicek-core-v2)

- **loro-crdt** — CRDT implementation for document synchronization
- **Vitest** — Unit testing framework

3.2.2 React Integration (mydenicek-react-v2)

- **React 19** — UI framework with modern features
- **React Context** — State management for document and sync state

3.2.3 Web Application (mywebnicek)

- **Vite** (rolldown-vite) — Build tool and development server
- **Fluent UI** (`@fluentui/react-components`) — Microsoft's design system
- **Playwright** — End-to-end testing

3.2.4 Sync Infrastructure

- **loro-websocket** — WebSocket adapter for Loro synchronization
- **Node.js** — Sync server runtime

3.3 Package Structure

```
mydenicek-monorepo/
  apps/
    mywebnicek/              # React web application
    mydenicek-sync-server/   # WebSocket sync server
  packages/
    mydenicek-core-v2/       # Core CRDT library
    mydenicek-react-v2/      # React hooks and context
    mydenicek-integration-tests/  # Cross-package tests
```

4 Evaluation

The evaluation strategy focuses on demonstrating that the CRDT-based approach can support the full range of *Denicek* programming experiences.

4.1 Functional correctness testing

- **Unit Tests**: Comprehensive tests for all core library operations
- **Conflict Resolution Tests**: Systematic tests verifying concurrent operation resolution
- **Integration Tests**: Tests verifying synchronization between multiple document instances

4.2 Performance evaluation

- **Benchmarks**: Measure operation latency, snapshot generation time, memory usage
- **Stress Tests**: Verify behavior under rapid operations and large documents

4.3 Formative example validation

Validate that the system supports the *Denicek* formative examples:

1. **Counter app**: Record increment action, replay on button click
2. **Todo app**: Record add-item action from input field
3. **Conference list**: Concurrent schema change and content addition
4. **Hello world**: Copy and apply formula to multiple items

5 Risks

5.1 Technical Risks

CRDT Library Limitations Loro may not support all required operations or may have performance issues.

- *Mitigation*: Early prototyping validated core operations. Performance testing will identify limits early.
- *Fallback*: Evaluate alternative CRDT libraries if critical issues arise.

Concurrent Wrap Semantics The deterministic wrapper ID approach may have edge cases.

- *Mitigation*: Comprehensive conflict resolution tests.
- *Fallback*: Fall back to random IDs with post-hoc conflict UI.

Recording/Replay Complexity Generalizing patches for replay may not handle all operation types.

- *Mitigation*: Start with simple operations, expand incrementally.
- *Fallback*: Limit replay to subset of operations.

5.2 Schedule Risks

Scope Creep The full *Denicek* feature set is extensive.

- *Mitigation*: Prioritize core operations and formative examples.
- *Priority order*: (1) Basic CRUD, (2) Wrap, (3) Sync, (4) Recording, (5) History view

6 Milestones / Deliverables

6.1 Primary Deliverables

MyDenicek Core Library (`@mydenicek/core-v2`) A fully functional TypeScript library providing document model, primitive operations, CRDT synchronization, undo/redo, recording/replay, and history/checkout.

MyWebnicek Application (`mywebnicek`) A demonstration web application with document renderer, editing controls, history view, and real-time collaboration.

Sync Server (`@mydenicek/sync-server`) A WebSocket server enabling multi-client document synchronization.

6.2 Secondary Deliverables

- Technical documentation (architecture, API reference, setup guides)
- Test suite (unit, integration, E2E tests, benchmarks)
- Research documentation comparing CRDT approach with original OT-based *Denicek*

6.3 Demo

A live demonstration will show document editing with multiple concurrent users, conflict resolution, recording/replay, and history navigation.

7 Time Schedule

The project spans November 2025 to July 2026 (8 months). Total estimated effort: approximately 180 man-days.

Milestone	**Time**	**Est. Days**
Analysis & Prototyping	Nov–Dec 2025	25
Core Library v1	Dec 2025–Jan 2026	30
Sync Infrastructure	Jan 2026	15
React Integration	Jan–Feb 2026	12
MyWebnicek UI	Feb–Mar 2026	25
Recording/Replay	Mar 2026	15
History View	Mar–Apr 2026	10
Testing & Validation	Apr 2026	15
Formative Examples	Apr–May 2026	10
Documentation	May–Jun 2026	12
Finalization	Jun–Jul 2026	10

Table 2: Project timeline and estimated effort

7.1 Progress as of January 2026

Work began in November 2025. Current status:

- Analysis and prototyping complete
- Core library v1 implemented (DenicekDocument, DenicekStore, DenicekModel)
- Sync infrastructure operational
- React integration complete
- MyWebnicek UI in progress
- History view recently added

8 Form of collaboration

8.1 Supervisor collaboration

Regular meetings with the supervisor Mgr. Tomáš Petříček, Ph.D. approximately every 2 weeks to review progress, discuss technical challenges, ensure alignment with research goals, and plan next steps. Ad-hoc communication via email for quick questions and clarifications.

8.2 Consultant collaboration

Jonathan Edwards (Denicek co-author) provides expertise on original *Denicek* design rationale, expected behavior of formative examples, and research positioning. Collaboration occurs through email and video calls as needed.

8.3 Repository management

- Primary repository: GitHub (krsion/MyDenicek)
- Version control: Git with feature branches
- Issue tracking: GitHub Issues
- Demo deployment: GitHub Pages

9 Presentation method

The project produces executable software demonstrated directly. The defense presentation includes:

1. **Live Demo**: Interactive demonstration of *MyWebnicek* showing document editing, real-time collaboration, conflict resolution, recording/replay, and history navigation
2. **Architecture Overview**: Presentation of layered architecture and key design decisions
3. **Research Findings**: Comparison of CRDT-based synchronization with OT
4. **Code Walkthrough**: Selected examples demonstrating key implementation details

References

[1] T. Petříček, et al. *Denicek: Computational Substrate for Document-Oriented End-User Programming.* In Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25). no. 32, pp. 1–19.

[2] N. Preguiça. *Conflict-free Replicated Data Types: An Overview.* 2018. `https://arxiv.org/abs/1806.10254`.

[3] M. D. Adams, et al. *Grove: A Bidirectionally Typed Collaborative Structure Editor Calculus.* ACM, 2024. DOI: `https://doi.org/10.1145/3704909`.

[4] M. Kleppmann, et al. *Local-first software: you own your data, in spite of the cloud.* Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. 2019.

[5] C. N. Klokmose, J. R. Eagan, and P. van Hardenberg. *MyWebstrates: Webstrates as Local-first Software.* In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24). Article 42, pp. 1–12.

[6] C. N. Klokmose, et al. *Webstrates: Shareable Dynamic Media.* In Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15). pp. 280–290.

[7] Loro. *Loro CRDT Library.* `https://loro.dev/`

[8] M. Kleppmann and A. R. Beresford. *A Conflict-Free Replicated JSON Datatype.* IEEE Transactions on Parallel and Distributed Systems, 2017.