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.
Expected start:	1.9.2025
Expected end:	31.3.2026

1 Introduction

MyDenicek is a CRDT-based local-first document editing substrate that enables collaborative editing of tree-structured documents with automatic conflict resolution. The project reimplements the synchronization layer of the original *Denicek* system [**?**], replacing Operational Transformation (OT) with Conflict-free Replicated Data Types (CRDTs) to satisfy the requirements of local-first software [**?**].

The original *Denicek* system provides end-user programming experiences including collaborative editing, programming by demonstration (recording and replaying user actions), incremental recomputation, and schema change control. However, its OT-based synchronization is complex, error-prone, and requires a central server. This project creates an alternative built on CRDTs that enables true peer-to-peer collaboration without a central authority.

MyWebnicek is a web-based programming system built on top of *MyDenicek*, providing a user interface for document navigation, editing, and collaborative features. It demonstrates the end-user programming capabilities through formative examples such as counter apps, todo lists, and document refactoring scenarios.

The project is inspired by *MyWebstrates* [**?**], a local-first CRDT-based alternative to Webstrates. Both *MyDenicek* (the core library) and *MyWebnicek* (the web application) are implemented in TypeScript.

1.1 Relation to Denicek

This project directly extends the research presented at *ACM UIST 2025* [**?**]. While the original Denicek uses Operational Transformation for synchronization, MyDenicek replaces this with a CRDT-based approach using Loro [**?**]. Loro provides a native movable tree CRDT (LoroTree) that handles concurrent structural edits, node moves, and reparenting automatically. The key architectural difference is that nodes are indexed by unique IDs rather than paths, avoiding the "shifting index" problem that occurs during concurrent structural edits.

The project maintains a clear separation between the CRDT substrate (`@mydenicek/core`) and the user interface (`mywebnicek`), enabling future development of alternative front-ends similar to how the original Denicek supports both Webnicek and Datnicek.

2 Project description

The project consists of two main components: the core CRDT library (*MyDenicek*) and the web application (*MyWebnicek*). The core library provides a TypeScript API for document manipulation with automatic conflict resolution, while the web application provides a user interface for end-user programming experiences.

2.1 User journey

The user journey through MyWebnicek consists of the following phases:

1. **Document Creation/Loading:** Users can create a new document or load an existing one. Documents can be shared via URL for collaborative editing.
2. **Document Navigation:** Users navigate the document tree using keyboard shortcuts or mouse clicks. The selected node is highlighted in both the rendered view and the JSON inspector.
3. **Document Editing:** Users perform edit operations through direct manipulation and toolbar controls. Operations include adding nodes, editing values, wrapping nodes, and deleting nodes.
4. **Recording/Replay:** Users can record a sequence of edit operations and replay them on different parts of the document, enabling programming by demonstration.
5. **Collaboration:** Multiple users can edit the same document simultaneously. Changes are synchronized via WebSocket and merged automatically using CRDTs.
6. **Undo/Redo:** Users can undo and redo operations. The undo history is local to each user.

2.2 Functional requirements

The following functional requirements specify the capabilities of the system. Requirements labeled [Core] relate to the `@mydenicek/core` library, while [Web] requirements relate to the `mywebnicek` application.

2.2.1 Core Library Requirements

FR-01: Document Representation [Core] The system shall represent documents as trees of nodes, where each node has a unique ID, a kind (element or value), and additional properties based on its kind.

FR-02: Element Nodes [Core] Element nodes shall have a tag name, a dictionary of attributes, and an ordered list of child node IDs.

FR-03: Value Nodes [Core] Value nodes shall contain a text string that can be modified through splice operations.

FR-04: Node Operations [Core] The system shall support the following operations on nodes:

- `addElementChildNode`: Add a new element child to a parent node
- `addValueChildNode`: Add a new value child to a parent node
- `deleteNode`: Remove a node and its descendants from the document
- `moveNode`: Move a node to a different parent or position
- `copyNode`: Create a copy of a node under a target parent
- `updateTag`: Change the tag name of an element node
- `updateAttribute`: Set or modify an attribute on an element node
- `spliceValue`: Insert, delete, or replace text within a value node
- `wrapNode`: Wrap a node in a new parent element

FR-05: Conflict Resolution [Core] The system shall automatically resolve conflicts when concurrent operations are applied:

- Concurrent wrap operations on the same node shall result in a single wrapper (using deterministic wrapper IDs)
- Concurrent attribute updates shall use Last-Writer-Wins (LWW) semantics
- Concurrent child additions shall preserve both children
- Concurrent tag updates shall use LWW semantics

FR-06: Undo/Redo [Core] The system shall maintain an undo stack and redo stack for each local session. Undo shall reverse the most recent local operation, and redo shall reapply the most recently undone operation.

FR-07: Recording [Core] The system shall support recording a sequence of operations starting from a specified node. Recorded operations shall use generalized node IDs (`$0`, `$1`, etc.) that can be bound to different nodes during replay.

FR-08: Replay [Core] The system shall support replaying a recorded script starting from a specified node. The generalized node IDs in the script shall be bound to actual node IDs based on the starting node.

FR-09: Transformations [Core] The system shall support defining transformations on element nodes that are automatically applied to new children. This enables pattern-based document manipulation.

2.2.2 Web Application Requirements

FR-10: Document Rendering [Web] The application shall render the document tree as HTML, allowing users to see the final document appearance.

FR-11: Node Selection [Web] Users shall be able to select nodes by clicking on them in the rendered view or navigating with keyboard shortcuts (arrow keys, Tab).

FR-12: JSON Inspector [Web] The application shall display a JSON view of the document structure, highlighting the currently selected node.

FR-13: Element Details Panel [Web] When an element node is selected, the application shall display its tag, attributes, and available operations.

FR-14: Add Node Interface [Web] Users shall be able to add child nodes through a popover interface that allows specifying the tag name or text content.

FR-15: Recording Controls [Web] The application shall provide controls to start and stop recording, and display the recorded script.

FR-16: Replay Controls [Web] The application shall provide controls to replay a recorded script on the currently selected node.

FR-17: Keyboard Shortcuts [Web] The application shall support keyboard shortcuts for common operations:

- Arrow keys: Navigate between nodes
- Delete/Backspace: Delete selected node
- Ctrl+Z: Undo
- Ctrl+Y / Ctrl+Shift+Z: Redo

FR-18: WebSocket Synchronization [Web] The application shall synchronize document changes with other clients via WebSocket connection to a sync server.

2.3 Non-functional requirements

The following non-functional requirements specify quality attributes and system constraints.

NFR-01: Convergence All replicas of a document shall eventually converge to the same state after receiving all operations, regardless of the order in which operations are received.

NFR-02: Offline Support The system shall support offline editing. Local changes shall be preserved and synchronized when connectivity is restored.

NFR-03: Response Time User interface operations (node selection, editing) shall complete within 100ms to provide responsive feedback.

NFR-04: Synchronization Latency Document changes shall be propagated to connected clients within 500ms under normal network conditions.

NFR-05: Browser Compatibility The web application shall function correctly on current versions of Chrome, Firefox, Safari, and Edge.

NFR-06: Type Safety The codebase shall use TypeScript with strict type checking. The use of `any` type is prohibited; `unknown` or specific types shall be used instead.

NFR-07: Code Quality The codebase shall pass ESLint checks with no errors. Unused imports shall be removed automatically, and imports shall be sorted consistently.

NFR-08: Test Coverage Core library functionality shall be covered by unit tests (Vitest). User interface functionality shall be covered by end-to-end tests (Playwright).

NFR-09: Documentation Public APIs shall be documented with JSDoc comments. The README shall contain setup instructions and usage examples.

NFR-10: Reproducibility The development environment shall be reproducible using npm. All dependencies shall be specified in package.json with locked versions.

2.4 Architecture

The system follows a monorepo structure with clear separation between packages:

```
apps/
  mywebnicek/               # React 19 + Fluent UI web application
  mydenicek-sync-server/    # WebSocket sync server
packages/
  mydenicek-core/           # Core CRDT logic (Loro wrapper)
  mydenicek-react/          # React hooks and context
```

2.4.1 Core Library Architecture

The core library (`@mydenicek/core`) provides three main classes:

DenicekStore is the entry point for document operations. It wraps a Loro document and provides:

- `modify(doc, updater)`: Execute a single change on the document
- `modifyTransaction(doc, updater)`: Execute multiple changes as a single undo step
- `undo(doc)` / `redo(doc)`: Undo/redo operations
- `startRecording(nodeId)` / `stopRecording()`: Recording controls
- `replay(doc, script, startNodeId)`: Replay a recorded script

DenicekModel provides read/write operations within `modify()` callbacks:

- Read: `getNode()`, `getAllNodes()`, `getParentId()`, `getChildren()`
- Write: `updateAttribute()`, `updateTag()`, `wrapNode()`, `deleteNode()`, `moveNode()`, `spliceValue()`, `addElementChildNo`
  `addValueChildNode()`, `copyNode()`

UndoManager maintains undo/redo stacks by capturing inverse patches for each operation.

2.4.2 Data Flow

1. User performs an action in the UI (e.g., clicks "Add Child")
2. React component calls `store.modify(doc, model => { ... })`
3. DenicekModel applies the change to the Loro document
4. Loro generates patches and notifies the UndoManager
5. If recording is active, patches are captured by the Recorder
6. Loro synchronizes changes to connected peers via WebSocket
7. Remote peers receive patches and update their local document
8. React re-renders based on the updated document state

3 Platform and technologies

The project uses the following technologies:

- **Programming Languages:**
  - TypeScript – all packages and applications
- **Frameworks and Libraries:**
  - React 19 – user interface framework
  - Fluent UI (`@fluentui/react-components`) – UI component library
  - Loro – CRDT library with native movable tree support
  - Vite – build tool and development server
- **Testing:**
  - Vitest – unit testing for core library
  - Playwright – end-to-end testing for web application
- **Code Quality:**
  - ESLint – linting with typescript-eslint
  - eslint-plugin-simple-import-sort – import sorting
  - eslint-plugin-unused-imports – unused import removal
- **Infrastructure:**
  - npm workspaces – monorepo management
  - GitHub Pages – deployment of web application
  - WebSocket – real-time synchronization

4 Evaluation

The evaluation approach focuses on demonstrating that the CRDT-based synchronization correctly implements the Denicek editing model and supports the formative examples from the original paper.

Unit Tests The core library includes unit tests (Vitest) that verify:

- Correctness of node operations (add, delete, move, copy, wrap)
- Conflict resolution behavior for concurrent operations
- Undo/redo functionality
- Recording and replay of operation sequences
- Synchronization between multiple documents

End-to-End Tests The web application includes Playwright tests that verify:

- User interface interactions (selection, navigation, editing)
- Recording and replay workflows

- Undo/redo through the UI
- Bulk operations on multiple nodes

Formative Examples The system will be evaluated against the formative examples from the Denicek paper:

- Counter app: Record incrementing a value and replay on button click
- Todo app: Record adding a list item and replay to add new todos
- Conference list: Concurrent editing with schema refactoring
- Hello world: Apply transformations across multiple items

Comparison with Original Denicek A qualitative comparison will assess:

- Feature parity with the original system
- Differences in conflict resolution behavior
- Developer ergonomics of the API

5 Risks

CRDT Limitations Some Denicek operations may not map cleanly to CRDT semantics. Mitigation: Identify problematic operations early and design workarounds or document limitations.

Loro Performance Large documents may cause performance issues. Mitigation: Implement lazy loading and pagination if needed; profile performance with realistic document sizes.

Conflict Resolution Semantics Users may expect different conflict resolution behavior than what CRDTs provide. Mitigation: Document the conflict resolution rules clearly; provide UI feedback when conflicts are resolved.

Browser Compatibility WebAssembly (used by Loro) may have compatibility issues in some browsers. Mitigation: Test on all target browsers; provide fallback or polyfills if needed.

Scope Creep The desire to implement all Denicek features may exceed available time. Mitigation: Prioritize core functionality (editing, sync, undo/redo) over advanced features (formula evaluation, debugging).

6 Milestones / Deliverables

By the project's conclusion, the following deliverables will be provided:

Functional MyDenicek Library A TypeScript library providing:

- CRDT-based document representation
- Complete set of edit operations
- Automatic conflict resolution
- Undo/redo support
- Recording and replay functionality

Functional MyWebnicek Application A React web application providing:

- Document rendering and navigation

- Visual editing interface
- Recording/replay UI
- Real-time collaboration via WebSocket

Technical Documentation Comprehensive documentation including:

- README with setup and usage instructions
- API documentation for the core library
- Architecture overview
- Conflict resolution rules

Test Suite Automated tests including:

- Unit tests for core library (Vitest)
- End-to-end tests for web application (Playwright)

Demo A live demonstration showing:

- Document editing and navigation
- Recording and replaying operations
- Real-time collaboration between multiple users
- Conflict resolution in action

Source Code The project will be available as an open-source repository on GitHub, with clear commit history and documentation.

7 Time Schedule

The project spans September 2025 to March 2026. The proposal was submitted on 16.10.2025 and approved on 11.11.2025. Work began with attendance at the DARE2025 summer school on CRDTs and local-first software, followed by analysis of the original Denicek system and prototyping. The following table shows completed work (DARE2025 + git history, counting each day with commits as 1 full working day) and estimated remaining effort.

The completed work includes migration from Automerge to Loro CRDT (January 2026), which provides better performance for tree-structured documents with its native movable tree CRDT. Remaining work focuses on real-time collaboration, UI polish, and demonstration of formative examples from the original Denicek paper.

8 Form of collaboration

Collaboration with the supervisor, Mgr. Tomáš Petříček, Ph.D., is essential for ensuring alignment with the original Denicek design and research objectives.

Milestone	Activity	Time	MD	Status
DARE2025	Summer school on CRDTs and local-first	Sep 2025	5	Done
Analysis	Study Denicek, CRDTs, prototype	Oct–Nov 2025	4	Done
Core + Undo	DenicekModel, UndoManager, basic UI	Nov–Dec 2025	14	Done
Recording/Replay	Recorder, generalized patches	Dec 2025	1	Done
Monorepo + Tests	Package structure, unit/E2E tests	Jan 2026	6	Done
Loro Migration	Replace Automerge with Loro CRDT	Jan 2026	4	Done
Completed			**34**	
Sync & Collaboration	WebSocket sync, multi-user testing	Jan–Feb 2026	8	Planned
Web Application Polish	UI improvements, conflict feedback	Feb 2026	6	Planned
Formative Examples	Counter, todo, conference list demos	Feb–Mar 2026	5	Planned
Testing & Bug Fixes	E2E tests, edge cases, bug fixes	Mar 2026	5	Planned
Documentation	README, API docs, video demo	Mar 2026	4	Planned
Finalization	Polish, presentation preparation	Mar 2026	3	Planned
Remaining (estimated)			**31**	
Total			**65**	

Table 1: Project timeline: completed work (DARE2025 summer school + git history) and estimated remaining effort.

8.1 Consulting plan

Regular Meetings Weekly meetings (approximately every week) with the supervisor, each lasting 30–60 minutes. Meetings cover:

- Progress updates on implementation
- Discussion of technical challenges
- Design decisions for conflict resolution
- Alignment with original Denicek semantics
- Planning for upcoming work

Ad-hoc Communication Email and messaging for quick questions and clarifications between meetings.

Code Reviews The supervisor may review code and provide feedback on architecture and implementation decisions.

8.2 Repository management

The project is hosted on GitHub at `https://github.com/krsion/MyDenicek`. Development follows standard Git workflow with feature branches and pull requests. The main branch is deployed to GitHub Pages for the live demo.

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] 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.

[3] Clemens Nylandsted Klokmose, James R. Eagan, and Peter van Hardenberg. 2024. MyWebstrates: Webstrates as Local-first Software. In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24). Association for Computing Machinery, New York, NY, USA, Article 42, 1–12.

[4] Loro: Reimagine state management with CRDTs. `https://loro.dev/`

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