Specification of team software project

Department of Software Engineering

Faculty of Mathematics and Physics, Charles University

Solvers: Bc. Ondrej Krsicka

Study program: Computer Science - Software and Data Engineering

Project title: Using CRDTs to enable collaborative editing in Denicek

Project type: Research project

Supervisor: Mgr. Tomas Petricek, Ph.D.

1 Introduction

Denicek [1] is a document-based end-user programming substrate that serves as the foundation for various programming systems. One such system is the web-based *Webnicek*, which enables users to create interactive HTML documents through end-user programming experiences including collaborative editing, programming by demonstration, and incremental recomputation.

The current Denicek synchronization layer relies on Operational Transformation (OT), which presents several challenges: it is complex, error-prone, and requires a central server for coordination. These limitations conflict with the principles of local-first software [4], which emphasize user data ownership and offline-first operation.

This project creates *MyDenicek*, a CRDT-based backend that replaces OT with Conflict-free Replicated Data Types using the Loro library [7]. The key innovation is adopting an **ID-based approach** instead of path-based addressing, which solves the "Shifting Index" problem that plagued OT-based systems during concurrent structural edits. *MyWebnicek* is a web-based programming system built on top of MyDenicek, providing functionality similar to Webnicek while demonstrating the capabilities of the new substrate.

1.1 Relation to other projects

This project directly builds upon the original Denicek system [1] and is inspired by *MyWebstrates* [5]: a local-first, CRDT-based alternative to Webstrates [6]. The naming convention follows this inspiration.

The project extends research on Conflict-free Replicated Data Types as summarized in Preguica's overview [2], and specifically leverages the Loro CRDT library [7] for its native tree support and efficient synchronization capabilities. The design was informed by Grove [3], a bidirectionally typed collaborative structure editor calculus, though the final implementation uses Loro's `LoroTree` for practical reasons including better move operation support and active maintenance.

Collaboration with the supervisor ensures alignment with the original Denicek design philosophy while exploring novel CRDT-based approaches to collaborative document editing.

2 Project description

The project delivers two main components: *MyDenicek*, a reusable TypeScript library providing CRDT-based document operations, and *MyWebnicek*, a React-based web application demonstrating the library's capabilities.

The key architectural decision is the adoption of **ID-based node identification** instead of path-based addressing. In path-based systems, concurrent structural edits cause paths to shift unpredictably. For example, if User A deletes the second child of a node while User B modifies the third child, path-based systems struggle to correctly identify the target of User B's operation. ID-based systems assign stable unique identifiers to each node, ensuring operations target the correct elements regardless of concurrent structural changes.

2.1 Functional requirements

The following functional requirements specify the capabilities of the MyDenicek core library and MyWebnicek application.

2.1.1 MyDenicek Core Library Requirements

FR-01: Document Representation The library shall represent HTML-like structured documents as trees where each node has a stable unique identifier, a kind (element or value), and associated data (tag and attributes for elements, text content for values).

FR-02: Node Operations The library shall expose an API for modifying documents through the following primitive operations:

- Add child node (element or value) to a parent at a specified index
- Add sibling node before or after a reference node
- Delete node and its subtree
- Update element tag name
- Update element attributes
- Edit value content with character-level text editing (splice operation)
- Move node to a new parent or position
- Wrap node in a new parent element

FR-03: CRDT-Based Merging The library shall provide CRDT-based operations for merging divergent document versions without requiring a central coordination server. Concurrent operations shall converge to identical states across all replicas.

FR-04: Conflict Resolution The library shall implement deterministic conflict resolution:

- Concurrent wrap operations on the same node shall produce a single wrapper using deterministic ID generation (`w-${nodeId}`)
- Concurrent tag or value edits shall use Last-Writer-Wins (LWW) semantics based on Lamport timestamps
- Concurrent child additions shall preserve all children with ordering determined by fractional indexing

FR-05: Export and Import The library shall support exporting document state as binary snapshots or incremental updates, and importing these to reconstruct or synchronize documents.

FR-06: Undo and Redo The library shall provide undo and redo functionality with configurable maximum steps and merge intervals for grouping related operations.

FR-07: History Recording The library shall record document changes as generalized patches that can be replayed on different document contexts, enabling programming by demonstration workflows.

FR-08: Version Tracking The library shall track document versions using vector clocks (frontiers) and support temporal checkout to previous versions.

FR-09: Event Subscriptions The library shall provide subscription mechanisms for:

- Document state changes (for UI updates)
- Local updates (for synchronization)
- Patch recording (for history capture)

FR-10: WebSocket Synchronization The library shall support real-time synchronization via WebSocket connections to a sync server, with room-based document isolation.

2.1.2 MyWebnicek Application Requirements

FR-11: Document Rendering The application shall render the document tree as interactive HTML elements, dynamically creating React components based on node tags and attributes.

FR-12: Node Navigation The application shall provide keyboard and mouse-based navigation through the document tree, including:

- Click to select nodes
- Ctrl/Cmd+click for multi-select
- Shift+click for range selection
- Keyboard navigation (parent, first child, previous/next sibling)

FR-13: Document Actions Toolbar The application shall provide a toolbar for common operations:

- Undo/Redo with visual state indicators
- Add child nodes (element or value)
- Edit text content of value nodes
- Rename element tags
- Wrap elements in containers
- Delete selected nodes

FR-14: Element Details Panel The application shall display and allow editing of the selected node's attributes.

FR-15: Recorded Script View The application shall display recorded actions as a table showing:

- Action type and parameters

- Target node identifiers
- Creation badges for actions that create new nodes
- Dependency information between actions

FR-16: Script Replay The application shall support replaying recorded scripts on different target nodes, enabling programming by demonstration.

FR-17: Collaboration Features The application shall provide:

- Sync toggle to connect/disconnect from the sync server
- Share button to copy shareable links (room ID in URL hash)
- Visual indication of connection status and room ID

FR-18: JSON View The application shall provide a raw JSON inspector for debugging document state.

FR-19: Snapshot View The application shall support capturing document snapshots for temporal comparison.

2.2 Non-functional requirements

NFR-01: Type Safety All components shall be implemented in TypeScript with strict type checking. The core library shall not expose Loro-specific types in its public API.

NFR-02: Abstraction Layer The `DenicekDocument` class shall serve as the sole public interface to document operations, hiding CRDT implementation details. Applications shall interact only through this abstraction.

NFR-03: Performance Node lookup, parent lookup, and children retrieval shall operate in O(1) time complexity through internal indexing.

NFR-04: Sync Latency Real-time synchronization shall propagate changes between connected clients within 500ms under normal network conditions.

NFR-05: Offline Support The library architecture shall support offline editing with automatic reconciliation when connectivity is restored.

NFR-06: Browser Compatibility MyWebnicek shall function correctly on current versions of Chrome, Firefox, Safari, and Edge.

NFR-07: Modularity The system shall maintain clear separation between:

- Core library (`@mydenicek/core-v2`) - CRDT operations
- React bindings (`@mydenicek/react-v2`) - hooks and context
- Web application (`mywebnicek`) - UI components
- Sync server (`mydenicek-sync-server`) - WebSocket coordination

NFR-08: Testability The core library shall have comprehensive unit tests. Integration tests shall verify multi-client synchronization scenarios.

NFR-09: Documentation All public APIs shall be documented with TypeScript JSDoc comments.

2.3 Architecture

The system follows a layered architecture with clear separation of concerns.

2.3.1 Package Structure

The project is organized as a monorepo with the following packages:

- **@mydenicek/core-v2** - Core CRDT library wrapping Loro
- **@mydenicek/react-v2** - React hooks and context providers
- **@mydenicek/mcp** - Model Context Protocol tool definitions
- **mywebnicek** - React 19 web application with Fluent UI
- **mydenicek-sync-server** - Node.js WebSocket sync server
- **mydenicek-integration-tests** - End-to-end synchronization tests

2.3.2 Core Library Architecture

The core library centers on two main classes:

DenicekDocument - The primary public API providing:

```
class DenicekDocument {
  // State access
  getRootId(): string | null;
  getNode(id: string): NodeData | null;
  getChildIds(parentId: string): string[];
  getParentId(nodeId: string): string | null;

  // Mutations (via callback)
  change(callback: (model: DenicekModel) => void): void;

  // Undo/Redo
  undo(): void;
  redo(): void;

  // Sync
  export(mode: "snapshot" | "update"): Uint8Array;
  import(bytes: Uint8Array): void;
  connectToSync(url: string, roomId: string): void;

  // Subscriptions
  subscribe(callback: () => void): () => void;
  subscribePatches(callback: (patches) => void): () => void;
}
```

DenicekModel - Write operations available inside `change()` callbacks:

```
interface DenicekModel {
  // Read
```

```
  rootId: string | null;
  getNode(id: string): NodeData | null;

  // Write
  createRootNode(tag: string): string;
  addChildNode(parentId, kind, data, index?): string;
  deleteNode(id: string): void;
  updateTag(id: string, newTag: string): void;
  updateAttribute(id, key, value): void;
  spliceValue(id, index, deleteCount, insert): void;
  moveNode(nodeId, newParentId, index?): void;
  wrapNode(targetId, wrapperTag, wrapperId?): void;
}
```

2.3.3 Data Types

```
interface ElementNodeData {
  id: string;
  kind: "element";
  tag: string;
  attrs: Record<string, unknown>;
}

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

type NodeData = ElementNodeData | ValueNodeData;
```

2.3.4 React Integration

The React library provides:

DenicekProvider - Context provider wrapping the application:

```
<DenicekProvider
  initializer={(model) => model.createRootNode("div")}
  onChange={() => console.log("Document changed")}
>
  <App />
</DenicekProvider>
```

Hooks:

- `useDocumentState()` - Access document and version counter
- `useConnectivity()` - Sync connection management
- `useRecording()` - History recording and replay
- `useDocumentActions()` - Mutation helpers with undo/redo
- `useSelection()` - Node selection state management

2.3.5 Synchronization Architecture

The sync system uses a WebSocket-based architecture:

1. **Client-side**: `LoroWebsocketClient` manages connections and rooms
2. **Server-side**: `SimpleServer` from `loro-websocket/server` handles room management
3. **Protocol**: Loro's native sync protocol with incremental updates
4. **Persistence**: File-based storage with configurable save intervals

Room-based isolation ensures documents are synchronized only among clients connected to the same room. The URL hash contains the room ID for easy sharing.

2.3.6 Conflict Resolution Strategy

The system handles concurrent operations as follows:

Scenario	**Resolution**
Concurrent wrap operations	Single wrapper with ID `w-${nodeId}`, LWW for tag
Concurrent child additions	Both preserved with fractional index ordering
Concurrent tag renames	Last-Writer-Wins based on timestamp
Concurrent value edits	Last-Writer-Wins based on timestamp
Concurrent wrap + add child	Both succeed independently
Concurrent wrap + rename	Both succeed (correct node targeted)

3 Platform and technologies

The project uses the following technologies:

- **Programming Languages:**
  - TypeScript - all components for type safety
- **Core Libraries:**
  - Loro (v1.5+) - CRDT data structures with native tree support

  - loro-websocket (v0.1+) - WebSocket sync protocol
  - loro-adaptors (v0.1+) - Integration utilities

- **Frontend:**
  - React 19 - UI framework
  - Fluent UI (v9.72+) - Component library
  - Vite - Build tool

- **Backend:**
  - Node.js - Sync server runtime
  - WebSocket - Real-time communication

- **Testing:**
  - Vitest - Unit testing
  - Playwright - End-to-end testing

- **Build and Development:**
  - pnpm - Package management (monorepo workspaces)
  - TypeScript 5.0+ - Compilation and type checking

4 Evaluation

The evaluation demonstrates that the CRDT-based approach correctly implements the required end-user programming experiences from Denicek.

4.1 Correctness Testing

Unit Tests verify individual operations:

- Basic operations (create, add, delete, update)
- Export/import round-trips
- Subscription notifications
- Undo/redo behavior
- Version tracking accuracy

Integration Tests verify multi-client scenarios:

- Real sync server startup and shutdown
- Two-client change propagation
- Concurrent edit convergence
- Offline edit reconciliation

4.2 Formative Examples

The system is evaluated against the formative examples from the original Denicek paper:

Counter App: User creates a document with value 1, wraps it in a formula (1+1), records the process, and replays it on button click to produce a counter.

Todo App: Similar to counter, but recorded actions add text from an input field as a new list item.

Conference List: Alice refactors a list of speakers from comma-separated items into a table. Meanwhile, Bob adds a new list item. The merged result contains all speakers including Bob's addition.

Conference Budget: Building on the conference list, formulas depending on values are tracked during refactoring.

Hello World: User makes the first line of a list lowercase, then capitalizes the first letter, copies the formula, and applies it to the whole list.

4.3 Comparison with OT

The ID-based CRDT approach is compared against the original OT implementation:

Aspect	**OT (Original)**	**CRDT (MyDenicek)**
Central server	Required	Optional (peer-to-peer capable)
Complexity	High (transform functions)	Lower (merge semantics)
Offline support	Limited	Native
Structural edits	Error-prone	Stable (ID-based)
Implementation	Custom	Loro library

5 Risks

Loro API Stability The Loro library is actively developed and APIs may change. Mitigation: The abstraction layer isolates application code from Loro internals, limiting the impact of API changes to the core library.

Performance at Scale Large documents with many concurrent editors may experience performance degradation. Mitigation: Loro is designed for efficiency, and the architecture supports future optimization through lazy loading and partial replication.

Feature Parity Some Denicek features may be difficult to map to CRDT semantics. Mitigation: Regular consultation with the supervisor and Denicek co-author ensures alignment on acceptable tradeoffs.

WebSocket Reliability Network instability may cause synchronization issues. Mitigation: The system supports offline editing with automatic reconciliation, and the sync server implements graceful reconnection handling.

Browser Compatibility Different browsers may handle WebSocket connections differently. Mitigation: Testing across major browsers and fallback mechanisms for connection issues.

6 Milestones / Deliverables

MyDenicek Core Library A fully functional TypeScript library (`@mydenicek/core-v2`) providing:

- CRDT-based document representation using Loro
- Complete set of document operations
- Undo/redo functionality
- History recording and replay
- WebSocket synchronization support
- Comprehensive unit tests

MyDenicek React Library React bindings (`@mydenicek/react-v2`) providing:

- Context provider for document state
- Hooks for document access and mutation
- Selection state management
- Connectivity management for sync

MyWebnicek Application A React web application demonstrating:

- Interactive document rendering
- Node navigation and selection
- Document editing toolbar
- Script recording and replay
- Real-time collaboration via sharing

Sync Server A Node.js WebSocket server providing:

- Room-based document synchronization
- Persistent storage of document state
- Logging for debugging and monitoring

Technical Documentation Comprehensive documentation including:

- API reference for all public interfaces
- Architecture documentation
- Setup and deployment instructions
- Evaluation results and comparison with OT

Demo A live demonstration showcasing:

- Document creation and editing
- Real-time collaboration between multiple clients
- Script recording and replay
- Conflict resolution in action

7 Time Schedule

The project spans November 2025 to July 2026 (8 months).

Phase	Activity	Timeline
Task 1	Study materials, analysis, prototyping - Literature review (CRDTs, Loro, Grove) - Prototype document representation - Evaluate CRDT approaches	Month 1-3
Task 2	Production implementation - MyDenicek core library - React bindings - MyWebnicek application - Sync server	Month 4-7
Task 3	Documentation - Technical report - Video demonstration - API documentation	Month 8
Task 4	Testing and evaluation - Integration tests - Formative example validation - Performance evaluation	Month 8

8 Form of collaboration

8.1 Consulting plan

Regular Meetings Bi-weekly meetings with the supervisor to discuss progress, technical challenges, and alignment with research objectives. Each meeting reviews completed work, addresses blockers, and plans next steps.

Design Collaboration The system design is developed in collaboration between the supervisor, the student, and Jonathan Edwards (Denicek co-author). This ensures the CRDT-based approach maintains compatibility with Denicek's design philosophy while exploring novel approaches.

Ad-hoc Communication Email and messaging for quick questions, code reviews, and urgent technical issues.

8.2 Repository Management

The project is maintained in a Git repository with:

- Main branch for stable releases
- Feature branches for development
- Regular commits with descriptive messages
- Code review before merging significant changes

References

[1] T. Petricek, 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. Preguica. *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). Association for Computing Machinery, 2024.

[6] C. N. Klokmose, J. R. Eagan, S. Baader, W. Mackay, and M. Beaudouin-Lafon. *Webstrates: Shareable Dynamic Media.* In Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15). Association for Computing Machinery, 2015.

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

[8] A. Cypher, et al. (eds.) *Watch what I do: programming by demonstration.* MIT Press, 1993.

[9] J. Edwards, et al. *Schema Evolution in Interactive Programming Systems.* The Art, Science, and Engineering of Programming, 9(1), 2-1. 2024.