The CodeVideo Framework: Event Sourcing for IDE State Representation in Educational Software

Full Stack Craft LLC

November 12, 2024

Contents

1 Introduction

The rapid growth of online education has transformed the software learning experience, creating unprecedented demand for high-quality, interactive software courses. As instructional video content rises, so do the demands for editing, structuring, and delivering this content at scale—a process that is both resource-intensive and technically challenging.

In this document, we present *CodeVideo*, a event-sourcing-based framework designed to streamline the creation and management of software instructional videos. By simulate every interaction (keystrokes and or mouse interactions) in an Integrated Development Environment (IDE), the framework can produce deterministic, repeatable sequences that simulate a live coding session. The result is not only the ability to generate a static snapshot of the IDE state at any given time but also the capability to generate entire instructional videos programmatically, ensuring consistency and precision across sessions.

2 Framework Architecture and Abstractions

The CodeVideo Framework represents a modular and highly abstracted approach to IDE-based video course creation. This section provides an in-depth look at its primary abstractions and the structured hierarchy that enables reproducible, fine-grained state control over IDE interactions.

2.1 Abstraction Layers

The framework is built on several core abstractions, each of which represents a fundamental unit of interaction within the software course environment.

2.1.1 Course Structure

At the topmost level of the *CodeVideo* framework, we define a course, represented by:

- A unique identifier, C_{id}
- A descriptive name, C_{name}
- A sequence of instructional units or lessons, denoted $L = \{L_1, L_2, \ldots, L_n\}$

2.1.2 Lesson Composition

Each lesson L is composed of:

- An initial state snapshot of the project workspace, $S_{initial}$
- A sequence of discrete actions $A = \{a_1, a_2, \ldots, a_m\}$
- A final state snapshot S_{final} that serves as the baseline for subsequent lessons

2.2 Action Representation

Central to the framework is the concept of an *action*, the smallest unit of interaction within the IDE. Actions, represented as a_i, encode the following elements:

$$a_i : \text{action\_type} \rightarrow \text{action\_value}$$

Examples of action types include text entry commands, cursor movements, file modifications, and verbal narration cues. For instance:

- **Text Entry Action:**

$$a_{type} = \text{“enter-text”}$$
$$a_{value} = \text{“console.log('Hello, world!');”}$$

- **Narration Action:**

$$a_{type} = \text{“speak-during”}$$
$$a_{value} = \text{“I'm logging a message to the console.”}$$

3 Snapshots: Capturing IDE State

Snapshots represent the entire state of the IDE at a given point, allowing for precise rollback and reproducibility:

$$\begin{aligned} S = \langle & \text{metadata}, \\ & \text{file\_structure}, \\ & \text{selected\_file}, \\ & \text{open\_files}, \\ & \text{editor\_content}, \\ & \text{caret\_position}, \\ & \text{terminal\_content} \rangle \end{aligned}$$

Where:

- **Metadata** contains high-level course information.

- **File Structure** denotes the directory tree visible in the IDE.
- **Selected File** and **Open Files** track active files.
- **Editor Content** and **Caret Position** record the current text and cursor position.
- **Terminal Content** displays command-line output.

4 Virtual Editor and Rendering Engine

The CodeVideo framework integrates a *virtual editor*, which emulates IDE interactions in a standalone environment:

- The virtual editor provides snapshots on demand via $getProjectSnapshot()$, allowing course authors to generate real-time previews of each state.

The rendering engine, *CodeVideo Studio*, provides a visual representation within the IDE, enabling authors to edit actions dynamically and validate course accuracy.

5 Validation and Quality Assurance

Each lesson's continuity is verified through a *snapshot validator*:

$$\forall i \in [1, n-1], \quad S_{final}(L_i) = S_{initial}(L_{i+1})$$

If any two consecutive snapshots do not match, the validator throws an error, ensuring seamless progression across lessons.

6 Recording and Playback Mechanisms

The framework supports a range of recorders, each capturing specific aspects of the user's IDE interaction:

- Keystrokes and cursor movements are recorded in the main editor.
- Interactions within the file tree, terminal, and narration windows are recorded and transformed to action format to build the repeatable and full IDE state representation.

7 Driver Integration for Automated Video Generation

CodeVideo provides multiple *drivers* for automated video generation:

- **CodeVideo Studio** our main editor, based on a Monaco editor along side an action editor. Includes time travel and snapshot validation.
- **Monaco Editor Driver** for localhost single editor-based video generation (beta)
- **VSCode Driver** for localhost complete IDE editor-based video generation (beta)

8 Future Directions: Advanced Abstract Syntax Tree Manipulations Through Time

Ongoing research aims to enable abstract syntax tree (AST) manipulations through time, which will allow modifications to past states within a lesson—such as renaming variables or functions—and propagating these changes in the project forward.

9 Conclusion

The *CodeVideo* framework offers a robust, reproducible method for generating IDE-based educational content, reducing manual editing and increasing course consistency. With its event-sourcing architecture, the framework not only captures precise IDE states but also provides a platform for creating interactive, context-rich programming tutorials.