Pontifícia Universidade Católica
do Rio de Janeiro

Paulo Victor Borges Oliveira Lima

Enhancing NCL Authoring through Visual Interfaces and Generative AI Agents

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática, do Departamento de Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Sérgio Colcher

Rio de Janeiro
August 2025

Pontifícia Universidade Católica
do Rio de Janeiro

Paulo Victor Borges Oliveira Lima

Enhancing NCL Authoring through Visual Interfaces and Generative AI Agents

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Prof. Sérgio Colcher
Advisor
Departamento de Informática – PUC-Rio

Profª. Simone Diniz Junqueira Barbosa
Departamento de Informática – PUC-Rio

Profª. Juliana Arriel
Departamento de Informática – PUC-Rio

Profª. Débora Christina Muchaluat Saade
Universidade Federal Fluminense

Prof. Carlos de Salles Soares Neto
Universidade Federal do Maranhão

Rio de Janeiro, August 7th, 2025

Paulo Victor Borges Oliveira Lima

Holds a bachelor's degree in Computer Science from the Federal University of Maranhão (UFMA).

Bibliographic data

Lima, Paulo Victor Borges Oliveira

Enhancing NCL Authoring through Visual Interfaces and Generative AI Agents / Paulo Victor Borges Oliveira Lima; advisor: Sérgio Colcher. – 2025.

88 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2025.

Inclui bibliografia

1. Informática – Teses. 2. Nested Context Language. 3. Autoria Visual. 4. Autoria Multimodal. 5. Inteligência Artificial Generativa. 6. Agentes de IA. I. Colcher, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Aos meus pais, irmãos e família
pelo apoio e encorajamento.

Acknowledgments

This work represents more than the completion of a master's degree, it is the materialization of a dream.

First and foremost, I would like to thank God for guiding and sustaining me through every moment of this journey.

To my parents, for their example, education, and constant affection. Mom, Dad, my deepest gratitude. There are no words sufficient to express how important you have been throughout this journey.

To my siblings, for all their love and for always being present with support, affection, and words of encouragement.

To my girlfriend, Ana Carolina, who never let love be lacking and made everything seem simpler. Thank you for your care, patience, and for being by my side in every stage of this journey.

To my grandmother Naly, who always encouraged me to read, to study, and to better understand the Portuguese language, her encouragement was so great that I ended up learning several other languages, though in this case, programming languages.

To my advisor, Professor Sérgio Colcher, for all the teachings, the support, and for believing in me from the very beginning.

To my colleagues at the TeleMídia Laboratory, for the knowledge exchange, partnership, and daily camaraderie. Especially to Daniel: thank you for always being available for a conversation or a helping hand. I am certain that much of the quality of my work comes from what I learned from you.

To the professors and colleagues at the Federal University of Maranhão, where I began my academic journey. In particular, I thank Professor Carlos Salles for his support and for the opportunities he presented me, and my lab colleague João Victor Gonçalves, whose partnership and encouragement were fundamental for me to believe in the path that brought me here.

To the professors who make up the Examination Committee, for their valuable contributions to this work.

To PUC-Rio, for the teachings and the excellent academic environment, without which this research could not have been carried out.

To all the professors and staff of the Department of Informatics at PUC-Rio, for their teachings and support throughout the master's program.

To the coffee producers from the countryside of Minas Gerais, whose beans resulted in the roughly 20 kilograms of specialty coffee consumed during the long hours dedicated to this research. The energy and motivation sustained throughout this journey are largely thanks to you.

To the friends and family who, in different ways, supported, encouraged, and contributed to the completion of this stage.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Lima, Paulo Victor Borges Oliveira; Colcher, Sérgio (Advisor). **Enhancing NCL Authoring through Visual Interfaces and Generative AI Agents**. Rio de Janeiro, 2025. 88p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The NCL (Nested Context Language) is widely used in the Brazilian Digital TV System for creating complex interactive applications. However, NCL document authoring presents significant challenges due to the inherent complexity of spatial layout definition and the need for specialized technical knowledge. This work proposes an innovative approach to simplify NCL authoring through the development of intuitive visual interfaces and generative artificial intelligence agents. The system is implemented as Visual Studio Code extensions and the effectiveness is evaluated through observational studies and usability assessment methodologies. The study compares traditional NCL authoring methods with the proposed tools. Preliminary results indicate a positive impact on usability, efficiency, and user satisfaction.

Keywords

Nested Context Language; Visual Authoring; Multimodal Authoring; Generative Artificial Intelligence; AI Agents.

Resumo

Lima, Paulo Victor Borges Oliveira; Colcher, Sérgio. **Facilitando a Autoria NCL através de Interfaces Visuais e Agentes de IA Generativa**. Rio de Janeiro, 2025. 88p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A linguagem NCL (Nested Context Language) é amplamente utilizada no Sistema Brasileiro de TV Digital para a criação de aplicações interativas complexas. No entanto, a autoria de documentos NCL apresenta desafios significativos devido à complexidade inerente à definição de layout espacial e à necessidade de conhecimento técnico especializado. Este trabalho propõe uma abordagem inovadora para simplificar a autoria em NCL por meio do desenvolvimento de interfaces visuais intuitivas e agentes de inteligência artificial generativa. O sistema é implementado como extensões do Visual Studio Code e sua efetividade é avaliada por meio de estudos observacionais e metodologias de avaliação de usabilidade. O estudo compara métodos tradicionais de autoria em NCL com as ferramentas propostas. Resultados preliminares indicam um impacto positivo na usabilidade, eficiência e satisfação dos usuários.

Palavras-chave

Nested Context Language; Autoria Visual; Autoria Multimodal; Inteligência Artificial Generativa; Agentes de IA.

Table of contents

List of figures

List of tables

List of Abreviations

ADI – Análise Digital de Imagens

AHM – Amsterdam Hypermedia Model

API – Application Programming Interface

CLT – Cognitive Load Theory

DDD – Domain-Driven Design

DTV – Digital Television

GPT – Generative Pre-trained Transformer

HCI – Human-Computer Interaction

IPTV – Internet Protocol Television

ITU-T – International Telecommunication Union - Telecommunication Standardization Sector

LLM – Large Language Model

NCL – Nested Context Language

NUI – Natural User Interface

RNN – Recurrent Neural Network

SBTVD – Sistema Brasileiro de Televisão Digital

SMIL – Synchronized Multimedia Integration Language

URI – Uniform Resource Identifier

VS Code – Visual Studio Code

XHTML – Extensible Hypertext Markup Language

If I have seen further, it is by standing on
the shoulders of giants.

Isaac Newton, *Letter to Robert Hooke, 1675.*

1
Introduction

1.1
Contextualization

The evolution of television in Brazil showcases the constant enhancement of signal transmission and reception technology. Until the early 2000s, analog TV was the dominant format, marking the first significant phase with its characteristic limitations in image and sound quality.

A significant leap forward came with the introduction of digital TV through the implementation of the Brazilian Digital Terrestrial Television System (SBTVD) in 2007. This transition brought substantial improvements, including high-definition (HD) broadcasting, surround sound, and the capability for multiprogramming.

Beyond these initial enhancements, in the subsequent years, advancements in digital TV have introduced a variety of interactive features that move beyond traditional passive viewing. Viewers can now access on-demand content, participate in live polls, engage with interactive advertisements, and receive real-time updates. These capabilities foster a more immersive and engaging experience tailored to the diverse preferences of the modern audience. As technology continues to advance, the demand for new transmission methods increases, which is essential for supporting emerging ideas and future innovations in broadcasting. The landscape of television has undergone a profound transformation over the past decades, evolving from traditional broadcast systems to sophisticated interactive digital platforms. This evolution has culminated in the emergence of TV 3.0 (SBTVD, 2024; MORENO et al., 2023), where viewers are no longer passive consumers but active participants in rich multimedia experiences. Digital TV (DTV) applications (ROCHA, 2013) exemplify this transformation, offering interactive experiences that seamlessly integrate text, images, audio, and video while responding dynamically to user inputs and adapting to diverse viewing contexts. However, creating these sophisticated multimedia applications presents significant technical challenges that limit their widespread adoption and development efficiency.

DTV applications, along with other interactive multimedia systems, are typically built using structured content known as *multimedia/hypermedia documents*. The standardization of these document structures has been a critical focus in the field, leading to the development of declarative languages such as the

Nested Context Language (NCL) (SOARES; RODRIGUES, 2006; SOARES et al., 2007; SOARES et al., 2009). NCL has gained recognition, being widely adopted in digital television middleware, such as Ginga, and standardized as an ITU-T Recommendation for IPTV services (SOARES; MORENO, 2014), making it a cornerstone technology for interactive multimedia development.

1.2 Motivation

The authoring of NCL documents presents considerable challenges that extend beyond mastering basic syntax. Developers must carefully define regions, descriptors, media objects, connectors, links, and logical relationships between elements, all while adhering to strict syntactic and semantic rules. Errors in these components, such as incorrect temporal relationships, invalid attribute configurations, or inconsistent element references, can severely compromise the functionality of multimedia applications (MORENO et al., 2010). The precision demanded by the development process requires substantial expertise and meticulous attention to detail, creating barriers that inhibit NCL adoption among developers with varying skill levels.

Traditional authoring tools for multimedia content creation have relied primarily on conventional mouse and keyboard interactions, which are effective in many scenarios but leave substantial room for improvement. While some research has explored Natural User Interfaces (NUIs) (KAUSHIK; JAIN, 2014; JAIN; LUND; WIXON, 2011), including body movements, gestures, and voice commands, for various applications such as smart home control (BHUIYAN; PICKING, 2009), their integration into multimedia authoring environments remains limited.

The emergence of Large Language Models (LLMs) has revolutionized numerous fields, enabling the development of sophisticated applications ranging from conversational agents, such as ChatGPT and Gemini, to complex reasoning systems (LI et al., 2024; TALEBIRAD; NADIRI, 2023; XI et al., 2025). Modern LLMs, including GPT-4o and DeepSeek V3, have demonstrated remarkable capabilities in natural language understanding, generation, and complex reasoning tasks. These models have shown particularly promising results in programming and code synthesis for mainstream programming languages (CHEN et al., 2021; CHEN et al., 2023; LI et al., 2022; NIJKAMP et al., 2022), offering potential benefits such as automated code generation, contextual suggestions, and enhanced error correction through iterative refinement.

However, when applied to domain-specific languages like NCL, LLMs en-

counter significant limitations (MORAES et al., 2023). Unlike mainstream programming languages with extensive online presence and community resources, NCL and similar specialized languages suffer from limited training data availability. This scarcity leads to reduced accuracy and increased hallucination, the generation of plausible yet incorrect information, when generating code for such specialized domains. Common challenges include syntactic inconsistencies, insufficient reasoning capabilities, and frequent generation of inaccurate or incomplete outputs (YERRAMILLI; VARMA; DWARAKANATH, 2021). These issues are particularly problematic in multimedia applications, where even minor errors can disrupt synchronization and interactivity.

Despite advancements in AI-driven development tools, a significant gap remains in the availability of integrated authoring environments specifically tailored for NCL and similar domain-specific multimedia languages. Current NCL authoring tools typically provide only basic syntax validation and editing support, with limited capabilities for advanced code generation, intelligent debugging, and contextual assistance. This lack of integrated, intelligent support could contribute to increased complexity, longer development times, and lower adoption among developers at all skill levels. Furthermore, NCL's adoption is hindered by the limited availability of community-driven resources, standardized best practices, and educational materials. Unlike mainstream programming languages, NCL lacks a robust ecosystem of tutorials, libraries, and community forums, significantly increasing the learning curve for new developers. This gap underscores the need for tools that not only streamline the authoring process but also provide educational support through contextual explanations, examples, and adaptive learning resources.

To address these challenges, multi-agent LLM architectures emerge as a promising solution (LI et al., 2024). By leveraging principles such as task decomposition (KHOT et al., 2022), iterative reflection, and automated error detection, these architectures can systematically improve the authoring process for complex domain-specific languages (LIU et al., 2023; ZHOU et al., 2022). Task decomposition is particularly well-suited to NCL's well-defined structure, enabling systems to divide complex authoring tasks into manageable subtasks, such as generating regions, defining descriptors, creating media objects, establishing connectors, linking components, and validating syntax. With each agent specializing in specific language aspects, document-wide consistency is maintained.

1.3 Proposal and Objectives

Within this context, this research aims to design, implement, and evaluate a two-component authoring tool for the NCL language. The tool is implemented as a Visual Studio Code extension and integrates two complementary authoring approaches: a multi-agent system that leverages large language models for conversational code generation, and an interactive visual interface for the direct manipulation of NCL region structures and live code preview.

To achieve this, the following specific objectives are defined:

1. Design a software architecture that integrates conversational AI and visual editing components, ensuring they operate on a synchronized document state.

2. Implement the multi-agent API for interpreting natural language commands and generating valid NCL code.

3. Develop an interactive visual interface for generating and editing NCL elements through direct manipulation.

4. Conduct an evaluation with developers to assess the tool's usability and its impact on the authoring workflow.

1.3.1 Text Structure

This work is organized as follows: Chapter 2 discusses theoretical foundations on hypermedia authoring tools and LLM-based code generation systems. Chapter 3 presents and discusses related work, reviewing existing hypermedia authoring tools and code-generation systems based on large language models. Chapter 4 details the proposed tool architecture, including agent responsibilities, prompt strategies, validation and correction mechanisms, and implementation details. Chapter 5 outlines the design of the observational study conducted to assess the tool's usability, effectiveness, and workflow integration and reports and analyzes the findings from the experiment. Chapter 6 concludes the dissertation and suggests directions for future work.

2
Theoretical Background

This chapter establishes the theoretical and technological foundation necessary to understand the proposed approach to intelligent hypermedia authoring. It begins by exploring the characteristics of authoring in hypermedia systems. The discussion then shifts to NCL. Subsequently, we analyze existing authoring tools and their roles within content creation workflows. The chapter concludes by examining the growing influence of large language models in intelligent systems, their capabilities as autonomous agents, and the theoretical underpinnings of prompt engineering that enable effective human-AI collaboration in domain-specific contexts.

2.1
Hypermedia Authoring

The concept of authoring in hypermedia systems represents a fundamental departure from traditional linear media creation, encompassing the design and development of interactive multimedia documents that integrate multiple media types within complex navigational and temporal structures (BULTERMAN; HARDMAN, 2005). This section establishes the theoretical foundations that distinguish hypermedia authoring from conventional content creation processes.

Hypermedia authoring, as conceptualized in the literature, extends beyond the simple aggregation of multimedia elements to encompass the creation of coherent interactive experiences that unfold across multiple dimensions: spatial, temporal, and behavioral. Unlike traditional authoring processes that focus primarily on sequential content organization, hypermedia authoring requires authors to consider complex interdependencies between media elements, user interaction patterns, and system behaviors.

The authoring process in hypermedia systems involves three fundamental dimensions of design complexity. First, the *spatial dimension* concerns the arrangement and positioning of media elements within presentation spaces, requiring consideration of layout constraints, responsive design principles, and visual hierarchy. Second, the *temporal dimension* addresses the synchronization and sequencing of media elements over time, encompassing both deterministic timing relationships and user-driven interaction patterns. Finally, the *behavioral dimension* defines the interactive capabilities of the system, specifying how user actions trigger system responses and how different system compo-

nents communicate and coordinate their behaviors.

In this context, the theoretical foundation for understanding hypermedia structure draws from several influential models that have shaped the field's conceptual development. The Dexter Hypertext Reference Model (HALASZ; SCHWARTZ, 1994) provided early theoretical grounding by establishing a three-layer architecture that separates storage, within-component, and presentation concerns. This separation of concerns principle remains fundamental to contemporary hypermedia system design and continues to influence architectural decisions in modern authoring environments. For example, in an educational hypermedia authoring tool, the storage layer may manage XML-based representations of media elements and their relationships, the within-component layer handles interactive behaviors defined by the author (e.g., quiz logic, media triggers), and the presentation layer is responsible for rendering these components visually to the learner. This layered architecture allows designers to independently modify content, logic, or layout without affecting the entire system.

Building upon these foundations, the Amsterdam Hypermedia Model (AHM) (HARDMAN; BULTERMAN; ROSSUM, 1993) introduced temporal modeling capabilities that are particularly relevant to multimedia hypermedia systems. Unlike the Dexter Model, which primarily focused on structural components, the AHM extended hypermedia modeling by incorporating explicit temporal dimensions, enabling authors to define when and for how long media objects should appear or interact during a presentation. At the core of AHM's temporal modeling lies Allen's interval algebra (ALLEN, 1983), a formal framework for describing temporal relationships between time intervals, such as "before," "meets," "overlaps," "during," and "equals." This allows for precise specification of how multimedia components (e.g., videos, audio clips, animations) relate to each other in time. For instance, using AHM, an author can define that a narration audio track should begin "during" a video segment, or that an image should "follow" a text animation with no overlap. These constraints are not merely visual cues but formal rules that can be parsed, validated, and enforced by authoring tools to ensure consistency and predictability in playback behavior. By introducing this temporal formalism, the AHM laid the groundwork for intelligent hypermedia authoring systems capable of orchestrating complex multimedia timelines, essential for domains like e-learning, digital storytelling, and interactive broadcasting.

The evolution toward declarative hypermedia models represents a significant paradigm shift in how interactive multimedia applications are conceptualized and created. Earlier hypermedia systems typically required procedural

programming approaches, where authors had to write step-by-step instructions describing how the system should behave. In contrast, declarative languages like the Synchronized Multimedia Integration Language (SMIL) (BULTERMAN, 2018) and subsequently NCL allow authors to describe what they want to happen rather than how to make it happen. This declarative approach offers several fundamental advantages. First, it separates content specification from presentation logic, meaning that the same hypermedia content can be adapted to different devices or presentation contexts without requiring a complete redesign. Second, it enables more intuitive authoring processes, as authors can focus on describing the desired multimedia relationships and behaviors rather than implementing the technical mechanisms to achieve them. Finally, declarative specifications tend to be more maintainable and understandable, as they express the author's intent more directly than procedural code. For example, in a declarative approach, an author might specify that "when video A ends, start video B and display image C." rather than writing event handlers and timing codes to coordinate these actions. The underlying system handles the implementation details while the author focuses on the creative and structural aspects of the hypermedia experience.

Futhermore, the cognitive demands associated with hypermedia authoring have been rigorously investigated in the context of multimedia learning research. A comprehensive systematic literature review conducted by Mutlu-Bayraktar, Cosgun e Altan (2019), analyzing 94 studies, underscores that cognitive load management remains a persistent and multifaceted challenge in the design and use of multimedia learning environments. Drawing upon Cognitive Load Theory (CLT) (PAAS; RENKL; SWELLER, 2003), the review delineates how authors and users are simultaneously affected by intrinsic cognitive loads during the creation and interpretation of multimedia artifacts. The findings reveal that authoring complex multimedia systems often requires managing intricate information hierarchies, orchestrating temporally synchronized media elements, and anticipating diverse patterns of user interaction across spatial and temporal dimensions. These activities frequently lead to elevated levels of extraneous cognitive load, particularly in environments that lack adequate structural scaffolding or intuitive design mechanisms.

In this context, effective hypermedia authoring environments must be conceived not only as tools for technical composition, but also as cognitively-aware interfaces that strategically minimize unnecessary cognitive overhead while fostering conditions that enhance users' ability to engage in complex, creative, and pedagogically sound design processes. This entails the integration of interface elements that support signaling, modality balancing, and adaptive

scaffolding, thereby aligning authoring support mechanisms with empirically grounded principles of multimedia learning.

2.2 NCL: Theoretical Foundations and Language Design

The Nested Context Language (NCL) represents a declarative approach to hypermedia authoring, specifically designed for the creation of interactive multimedia applications in digital television and web-based environments (SOARES, 2009). Like other declarative languages, NCL is grounded in a conceptual data model that defines structural concepts, events, and the relationships between elements, along with the rules for data manipulation and updating. At the core of NCL lies the Nested Context Model (NCM), which provides the conceptual framework that differentiates NCL from simpler hypermedia models by enabling treatment of multimedia relationships and temporal synchronization. NCM extends traditional hypermedia concepts by incorporating powerful organizational and relationship modeling capabilities. NCM introduces a set of abstractions that underpin the design of NCL. Nodes represent both media and compositions; media nodes carry actual multimedia content such as videos, images, or texts, while composition nodes provide hierarchical structure. Each node is equipped with anchors, which are not embedded in the content but are structurally defined. These anchors allow fine-grained referencing to temporal segments of a video, spatial regions in an image, or specific samples in an audio stream, reflecting a separation between structure and content.

Complementing these elements are descriptors and properties, which together manage presentation attributes like color and position. Descriptors define how and where content should be presented, supporting adaptive presentations across different devices and contexts. This separation between content and presentation facilitates multi-platform deployment and interface flexibility.

Composition nodes, including context and switch nodes, enable hierarchical content structuring and conditional content selection. Context nodes allow nesting of other nodes, creating layered content organizations that are independent of timing or visual arrangement. Switch nodes, on the other hand, support adaptation by declaring sets of alternatives, with the system selecting among them based on rules tied to user preferences, device features, or environmental conditions. Ports play a crucial role in exposing internal node interfaces to external links, ensuring modularity and reuse while preserving encapsulation.

One of NCL's defining features is its robust model for specifying relationships among elements. Connectors abstract reusable relationship patterns by defining roles and how they interact. Links instantiate these patterns by associating specific node interfaces with the connector roles. In the context of digital TV, these connectors follow a causal model: when certain conditions are met, predefined actions are triggered. This event-condition-action logic provides a powerful way to describe complex synchronizations and user interactions.

The realization of NCM in NCL follows a structured XML-based syntax. NCL documents are organized into two main parts: a head section, where definitions such as descriptors, regions, and connectors reside; and a body section, where actual media elements and their relationships are instantiated. The body itself functions as a context node and may contain other nested contexts, media objects, and links.

Media elements refer to actual content files and declare their properties explicitly or by referencing descriptors. Context elements enable hierarchical structuring of related media, encapsulating interactions and organization. Fine-grained synchronization and interaction are achieved through area elements, which define spatial, temporal, or textual anchors. Ports allow context nodes to expose these internal interfaces, while properties may also be treated as addressable interfaces, enabling interactive control over presentation parameters such as position or volume.

NCL's spatial modeling capabilities are expressed through regions, which define nested spatial containers with relative or absolute positions. These regions are organized into a region base and can be reused across descriptors. Temporal and visual behaviors are specified through descriptors, which group parameters related to duration, layout, player behavior, and more. While NCL does not directly encode formal temporal relations such as those defined by Allen, its connector system supports equivalent logic declaratively.

Relationship modeling is expressed via causal connectors and links. A causal connector encapsulates a reusable event-condition-action pattern, which can be parameterized and reused across applications. A link then binds actual media interfaces to the connector's roles, establishing specific behaviors in the context of a document. This system promotes both reuse and modularity, while supporting rich interaction scenarios.

For adaptive content, NCL provides mechanisms such as the switch element for selecting among media options based on runtime conditions, and descriptorSwitch for selecting among presentation configurations. Rule evaluation, which supports both simple and compound logical conditions, governs these adaptive mechanisms, enabling context-aware multimedia presentation.

To support modular development and reuse, NCL organizes reusable components into logical bases, such as regionBase, descriptorBase, connectorBase, and ruleBase. These can be imported across documents using importBase, with aliases ensuring namespace isolation. Cross-document references further enhance modularity by enabling the construction of distributed applications composed from independent, reusable parts.

Altogether, NCL offers a powerful declarative model for expressing complex multimedia applications. Its design separates concerns cleanly—structure, presentation, and behavior—allowing developers to work independently on each aspect while maintaining cohesion. Reuse and modularity reduce development effort, while the declarative nature promotes clarity and maintainability.

Despite its expressiveness, the richness of NCL's specification model introduces authoring complexity. Understanding the mapping from NCM abstractions to NCL syntax, and managing the explicit declaration of interfaces, relationships, and rules, can be overwhelming—particularly in large-scale or adaptive applications. This tension between power and complexity underscores the motivation for intelligent authoring support, which this dissertation seeks to address. The next sections will explore how current tools support NCL authoring and outline the proposal for a multi-agent system that leverages NCL's structure to enhance the authoring experience.

This practical implementation also reveals the authoring challenges that motivate the research presented in this dissertation. The expressiveness that makes NCL powerful for applications also creates complexity barriers for authors, particularly in understanding the relationship between abstract NCM concepts and their concrete realization in NCL syntax. The need for explicit specification of all relationships, interfaces, and adaptation rules creates authoring overhead that can be overwhelming for complex applications, highlighting the opportunity for intelligent authoring assistance systems that can leverage NCL's declarative nature while reducing the burden on human authors.

These practical considerations establish the context for examining existing authoring tools and approaches in the following sections, and ultimately for developing the intelligent multi-agent authoring system that forms the core contribution of this research.

2.3 Theoretical Foundations of Authoring Tool Design

Authoring tools represent the critical interface between human creativity and technical implementation in hypermedia development. The design of

effective authoring tools requires careful consideration of human-computer interaction principles, cognitive psychology ideas, and software engineering best practices. This section examines the theoretical foundations that inform authoring tool design decisions and their application to hypermedia authoring contexts.

2.3.1 Software Engineering Perspectives on Authoring Systems

The development of robust authoring tools requires consideration of software engineering principles that ensure maintainability, extensibility, and reliability (GAMMA et al., 1995). The application of established software design patterns and architectural approaches to authoring tool development can significantly improve both the quality of the tools themselves and their ability to support complex authoring tasks.

Domain-Driven Design (DDD) (EVANS, 2003) provides a particularly valuable set of principles for structuring information systems, as it emphasizes aligning software architecture with the domain's conceptual foundations. A central principle of DDD is the creation of a ubiquitous language, a shared vocabulary used consistently by both developers and domain experts (e.g., multimedia authors, designers). This common language ensures that models, interfaces, and internal logic accurately reflect real-world authoring concepts, thereby reducing miscommunication and enhancing the expressiveness and clarity of the system. In the context of AI-assisted authoring, the benefits of a ubiquitous language extend even further. When generating prompts for intelligent agents or structuring value objects that encapsulate key domain concepts (such as media descriptors, temporal constraints, or layout regions), the use of a precise, shared vocabulary improves the accuracy of interpretations and the effectiveness of automated reasoning. Thus, DDD not only enhances the software's internal consistency but also supports seamless interaction between human authors and AI-based components. For example, in a multimodal authoring tool, a value object such as TemporalConstraint—defined using domain language like "startsAfter", "overlapsWith", or "endsWithDelay", can be consistently referenced both in the interface logic and AI-generated prompts. This shared vocabulary ensures that when a user issues a voice command such as "make the image appear after the video ends", the system's underlying domain model interprets and maps the intent accurately, reinforcing coherence between the author's conceptual model and the system's behavior.

Hexagonal Architecture (Ports and Adapters) (COCKBURN, 2005) offers a complementary architectural approach that reinforces the separation of

concerns by isolating the core domain logic from peripheral components such as user interfaces, file systems, and communication protocols. This pattern is especially relevant to the development of authoring tools, which must operate in heterogeneous environments and interact with a variety of external systems, including media repositories, version control platforms, validation services, and content delivery infrastructures. By structuring systems around interchangeable adapters, the hexagonal model provides a high degree of modularity and extensibility. Authoring tools built on this architecture can easily support multiple interaction modalities (e.g., graphical editors, code-based interfaces, voice or gesture inputs) without altering the domain logic. For instance, the same *GenerateCodeUseCase* may be reused across distinct contexts by routing requests through different adapters: one connected to an LLM API, another to a local rule engine, and a third to a fine-tuned domain-specific model. Similarly, input modalities like speech commands or multimodal gestures can be translated into standard domain instructions by dedicated interface adapters, preserving consistency and reducing coupling between system layers.

In essence, the hexagonal approach ensures that authoring tools remain resilient to technological shifts, adaptable to new modes of interaction and AI integration, and maintainable as generation strategies and infrastructure evolve, without requiring invasive changes to the underlying domain logic.

2.4 Large Language Models: Architectural and Theoretical Foundations

Large Language Models represent a significant paradigm shift in artificial intelligence, achieving remarkable performance in tasks of natural language understanding and generation (BROWN et al., 2020). Exploring their architectural and theoretical foundations is fundamental to appreciating their transformative potential for intelligent authoring systems, while also recognizing the challenges inherent in adapting these models to specialized domains.

At the core of most modern LLMs lies the transformer architecture (VASWANI et al., 2017), a groundbreaking framework that replaces traditional recurrent connections with a self-attention mechanism. Unlike recurrent neural networks, which process tokens sequentially and may struggle with long-term dependencies, transformers employ self-attention to evaluate relationships between all tokens in parallel, regardless of their distance in the sequence. This parallelism not only improves computational efficiency but also allows the model to capture intricate contextual relationships that would otherwise be diluted or lost over long spans of text.

The self-attention mechanism operates by first projecting the input

sequence into three distinct learned representations: queries (Q), keys (K), and values (V). The model then calculates an attention score for each pair of tokens by taking the scaled dot-product of their query and key representations. These scores are normalized using a softmax function to produce a distribution of attention weights, indicating how strongly each token should influence another. Finally, these weights are applied to the value vectors, aggregating the relevant contextual information in a way that dynamically adapts to the needs of each token's representation. This dynamic re-weighting allows the model to emphasize crucial information while suppressing less relevant context.

An extension of this mechanism, known as multi-head attention, further enriches the model's capacity to capture complex patterns. Instead of computing a single attention distribution, multi-head attention projects the input into multiple independent attention spaces, enabling the network to simultaneously focus on diverse linguistic attributes, for example, syntactic roles, semantic dependencies, or even positional structure. By concatenating and integrating the outputs of these parallel attention heads, the transformer can synthesize a far richer and more nuanced representation of the input sequence.

Overall, these architectural innovations have made it possible for LLMs to scale to billions of parameters, training on enormous text corpora to develop general-purpose language capabilities that can then be adapted to specific authoring contexts. Yet, this flexibility comes with significant challenges, such as ensuring domain adaptation, controlling hallucinations, and maintaining interpretability. Understanding these foundations is therefore crucial for designing intelligent authoring systems that can leverage LLMs both effectively and responsibly.

Despite their impressive capabilities, LLMs exhibit several important limitations that must be considered in their application to specialized domains. *Hallucination* (ZHANG et al., 2025; HUANG et al., 2025), the generation of plausible but factually incorrect information, represents a significant challenge for applications requiring high accuracy. This limitation is particularly problematic in technical domains where factual errors can lead to system failures or security vulnerabilities. *Context window limitations* constrain the amount of information that can be processed in a single interaction, limiting the model's ability to work with large documents or maintain context across extended conversations. While recent architectures have expanded context windows significantly, the computational cost grows quadratically with context length, creating practical limitations for real-world applications. *Training data bias* can lead to outputs that reflect biases present in the training corpus, potentially resulting in unfair or inappropriate responses in certain contexts (BEN-

DER et al., 2021). In technical applications, this can manifest as adherence to outdated practices or suboptimal design patterns that were prevalent in the training data.

2.5 LLM-Based Intelligent Agents: Theoretical Framework

The evolution of LLMs from passive text generation systems to active intelligent agents represents a significant advancement in artificial intelligence applications (XI et al., 2025). This transformation enables the development of systems that can plan, reason, and execute complex tasks through interaction with external environments and tools.

2.5.1 Agent Architecture and Cognitive Capabilities

LLM-based intelligent agents typically incorporate several key architectural components that enable autonomous operation in complex environments. The *planning component* enables agents to decompose complex tasks into sequences of manageable actions, reasoning about dependencies and constraints to develop effective execution strategies. This capability draws from classical AI planning research while leveraging the LLM's natural language understanding to work with more flexible and intuitive task specifications.

The *memory system* enables agents to maintain context and state across multiple interactions, supporting more coherent and purposeful behavior over extended periods. Different memory architectures have been proposed, ranging from simple context window management to external memory systems that can store and retrieve relevant information based on similarity or relevance metrics.

Tool integration capabilities enable agents to interact with external systems, APIs, and software tools to gather information and execute actions beyond text generation (SCHICK et al., 2023). This capability significantly expands the practical utility of LLM-based agents by enabling them to work with real-world systems and data sources. Furthermore, *Reflection and self-correction* mechanisms enable agents to evaluate their own outputs and make improvements based on feedback or validation results (SHINN et al., 2023). This capability is particularly important for applications requiring high reliability, as it enables the agent to detect and correct errors through iterative refinement.

2.5.2 Multi-Agent Systems and Collaborative Intelligence

Multi-agent systems extend the capabilities of individual LLM agents by enabling collaboration between specialized agents, each optimized for specific types of tasks or domains. This approach offers both theoretical and practical advantages over single-agent systems. One key advantage is task specialization, where different agents can develop expertise in specific domains or types of operations, achieving higher performance than generalist agents. In the context of hypermedia authoring, for example, specialized agents could focus on temporal modeling, spatial layout, content management, or validation, each optimized for its particular responsibility. Another important advantage is parallel processing, which allows multiple agents to address different aspects of a complex problem simultaneously, reducing overall completion time and enabling a wider range of problem-solving strategies. This capability is especially valuable for authoring workflows that involve multiple independent or loosely coupled subtasks. Furthermore, multi-agent systems can enhance robustness and fault tolerance through redundancy and cross-validation among agents. If one agent produces suboptimal results or fails entirely, other agents may compensate or offer alternative solutions, which is particularly relevant in scenarios demanding high reliability. Finally, multi-agent systems can give rise to emergent collective intelligence, where interactions between agents generate solutions that exceed the capabilities of any single agent.

2.6 Prompt Engineering: Theoretical Principles and Advanced Techniques

Prompt engineering represents the systematic design of inputs that guide LLM behavior toward desired outcomes. As the primary interface between human intentions and LLM capabilities, prompt engineering has evolved into a discipline with solid theoretical underpinnings and empirical best practices. The effectiveness of prompt engineering can be understood through the lens of cognitive science and linguistics (WEI et al., 2022a) since LLMs generally perform better when prompts align with natural human reasoning patterns and communication structures. This alignment suggests that prompt design should take into account how humans approach problem-solving and knowledge retrieval. Pragmatic considerations from linguistics also inform how context and implied meaning influence LLM responses, since the pragmatic interpretation of a prompt can significantly affect model behavior. Consequently, prompt designers must reflect on both the literal meaning of their instructions and the possible pragmatic implications or unintended

interpretations.

Several advanced techniques have been proposed to enhance LLM performance on reasoning and generation tasks (ZHOU et al., 2022). Among these, chain-of-thought prompting (WEI et al., 2022b) encourages models to articulate their reasoning in a step-by-step fashion, improving performance on complex problems by helping the model track intermediate results and progressively build reasoning. This technique essentially allows the model to treat its own generated text as an external form of memory or a reasoning scaffold. This flow is exemplified in Figure 2.1, which illustrates a typical chain of thoughts applied in contexts of choice and option evaluation.

Furthermore, building on this idea, tree-of-thought prompting (YAO et al., 2023) explores multiple reasoning branches in parallel, making it possible to examine a broader solution space and to choose the most promising reasoning path. Self-consistency approaches (WANG et al., 2022) further enhance reliability by generating multiple independent solutions and selecting the most consistent or frequent answer, thereby exploiting the statistical properties of the model's output to improve correctness when a definitive solution exists. This reasoning process is illustrated in Figure 2.2, which visually represents how different branches of thought can evolve and converge toward an optimal solution.

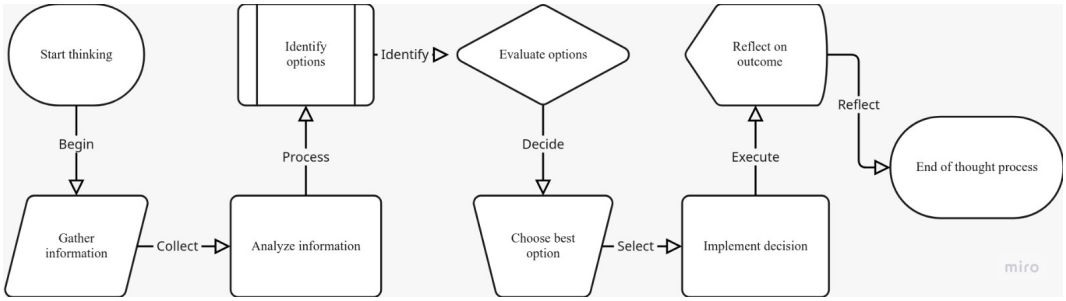

Figure 2.1: Example of a chain of thoughts representing the cognitive flow involved in the decision-making process.

Applying prompt engineering to specialized domains like software development and technical documentation introduces additional requirements and constraints (CHEN et al., 2021). Technical domains often demand higher levels of precision and adherence to formal specifications than ordinary language tasks. To meet these demands, prompts must emphasize technical accuracy, including syntactic correctness, semantic validity, and strict compliance with established standards. Domain knowledge should also be carefully integrated, incorporating relevant concepts, terminology, and best practices through examples, reference materials, or explicit instructions. Furthermore, prompts should encourage the model to include self-validation steps and checking procedures

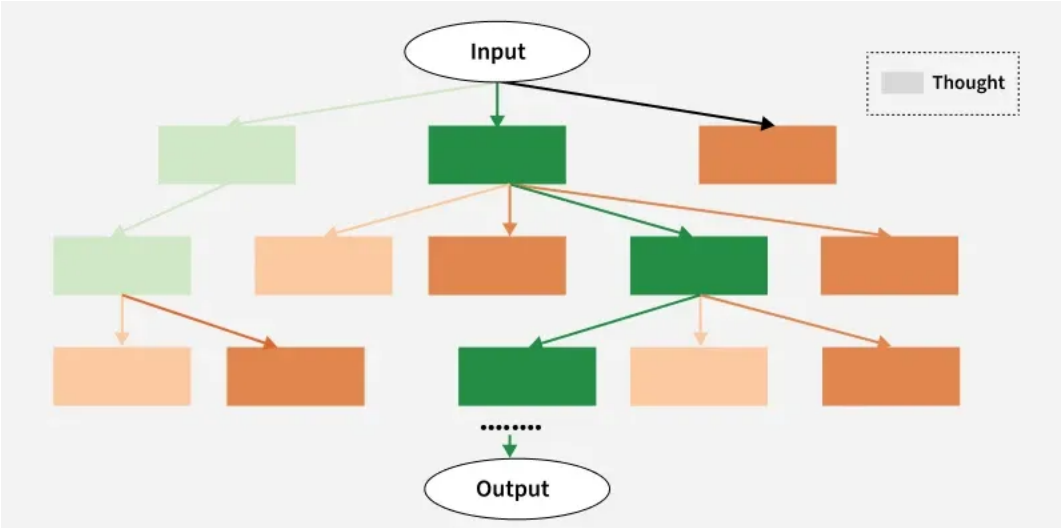

Figure 2.2: Example of the Tree of Thought method, where multiple branches of reasoning are explored in parallel, allowing the most promising line of thought to be selected

in its outputs, helping to identify inconsistencies before they propagate. Iterative refinement is also valuable, allowing initial results to be reviewed and improved over successive prompt rounds, which is critical for complex tasks that demand multi-stage problem-solving.

2.6.1 Considerations

In summary, this chapter has established the theoretical foundations for understanding the research problem and the proposed solution. The discussion of hypermedia authoring highlighted the cognitive and technical challenges arising from the multidimensional nature of hypermedia design, encompassing spatial, temporal, and behavioral complexities. The analysis of NCL showed how declarative approaches can mitigate some of these challenges while revealing a continued need for more accessible authoring tools. Considerations of authoring tool design principles underscored the importance of human-computer interaction, cognitive load management, and software engineering best practices in supporting effective development environments. The examination of Large Language Models clarified their capabilities and limitations, especially for specialized domain tasks. The discussion of LLM-based intelligent agents and multi-agent systems provided a conceptual basis for understanding how collaborative AI systems can address complex challenges through specialization and coordination. Finally, the study of prompt engineering established a framework for effectively guiding LLM behavior in these specialized contexts. Altogether, these theoretical foundations inform the methodological choices and

design decisions presented in the subsequent chapters, supporting the development of an intelligent multi-agent system for NCL authoring that addresses these challenges while leveraging the full potential of modern AI technologies.

3
Related Work

This chapter presents an analysis of existing research and tools related to intelligent hypermedia authoring systems. We examine the current state of the art across several key areas: authoring tools, NCL-specific authoring environments, validation systems, and AI-powered code generation approaches. Through systematic analysis and comparison, we identify the strengths and limitations of existing solutions, establishing a clear rationale for the research contributions presented in this dissertation.

The search strategy employed multiple complementary approaches to ensure comprehensive coverage of relevant literature. Primary searches were conducted using the following academic databases and digital libraries:

- **IEEE Xplore Digital Library:** Comprehensive coverage of computer science and engineering publications
- **ACM Digital Library:** Extensive collection of computing and information technology research
- **Google Scholar:** Broad coverage including conference proceedings, theses, and technical reports

Additionally, targeted searches were conducted within specific conference proceedings known to publish relevant research:

- WebMedia (Brazilian Symposium on Multimedia and the Web)
- ACM Multimedia Conference
- CHI Conference on Human Factors in Computing Systems
- UIST (User Interface Software and Technology)

The search terms employed were carefully constructed to capture relevant research across multiple related domains. Primary search strings included:

- ("NCL" OR "Nested Context Language") AND ("authoring" OR "development")
- "hypermedia authoring" AND ("tools" OR "systems" OR "environments")
- "multimedia authoring" AND ("declarative" OR "temporal modeling")
- ("LLM" OR "large language model") AND "code generation"

- "multi-agent systems" AND ("code generation" OR "software development")
- "domain-specific language" AND ("generation" OR "authoring" OR "AI")
- "intelligent authoring systems" AND ("multimedia" OR "hypermedia")
- "prompt engineering" AND ("code generation" OR "specialized domains")

The selection of relevant literature was guided by explicit inclusion and exclusion criteria designed to ensure focus on high-quality, relevant research while maintaining sufficient scope to capture the breadth of related work.

Inclusion Criteria

- Peer-reviewed publications in conferences, journals, or workshops
- Studies published between 2010–2025
- Research addressing authoring tools or systems for hypermedia, multimedia, or interactive applications
- Work investigating AI-based or machine-learning approaches to code generation
- Studies specifically examining NCL, or similar declarative multimedia languages
- Research on multi-agent systems applied to software development or content creation
- Theoretical or empirical work on prompt engineering for specialized domains
- Tools or systems demonstrating approaches to multimedia authoring workflows

Exclusion Criteria

- General multimedia editing tools without programmatic authoring capabilities
- Studies focused exclusively on content management without authoring considerations
- Research addressing unrelated programming languages or application domains
- Publications in non-peer-reviewed venues without demonstrated impact
- Work addressing only low-level multimedia processing without authoring abstractions
- Studies focused on hardware-specific implementations without generalizable insights
- Publications not available in English, Portuguese, or without accessible full-text

3.1 NCL-Specific Authoring Environments

The NCL ecosystem has spawned several specialized authoring tools, each embodying distinct approaches to managing the language's inherent complexity. These tools demonstrate various strategies for making NCL's powerful features accessible to developers, but as we will see, they often introduce trade-offs between usability, scalability, and expressive power.

NCL Composer (GUIMARÃES, 2007) showed a visual approach to NCL authoring through its innovative node-based interface. This tool abstracted NCL's textual complexity into designed visual metaphors, enabling users to define media objects, contexts, and relationships through direct graphical manipulation. By doing so, it demonstrated the potential of visual interfaces to improve accessibility and improve usability of authors, allowing them to focus on the creative and structural aspects of their hypermedia documents. The template-based paradigm found expression in **NEXT (NCL Editor Supporting XTemplate)** (MATTOS; SILVA; MUCHALUAT-SAADE, 2013), which leveraged hypermedia templates to reduce cognitive load. XTemplate enabled the definition and reuse of templates, streamlining development for recurring design patterns while still allowing the creation of new ones.

STEVE (Spatio-Temporal View Editor) (MATTOS; MUCHALUAT-SAADE, 2018) focused on the spatio-temporal aspects of

composition, combining timeline-based view, event-based editing, and spatial layout design. Despite its innovative interface, the system exhibited limitations in supporting complex interactive behaviors and failed to capture the *full expressive power* of NCL. More recent versions, such as Mattos et al. (2025), extend this paradigm by integrating artificial intelligence techniques to suggest new NCL elements, including sensory effects, thereby expanding the authoring possibilities.

The integration approach emerged through **NCL Eclipse** (AZEVEDO MARIO MEIRELES TEIXEIRA, 2009), an Eclipse IDE plugin providing features like syntax highlighting and code completion. This strategy catered to developers but suffered from the platform's declining popularity and, being purely textual, *excluded the visual design paradigms* essential for effective multimedia authoring.

Finally, **Lua2NCL** (MORAES et al., 2016) addressed NCL's verbosity by using Lua as an intermediate representation. While effective in reducing code length, it *required programming knowledge*, limiting accessibility, and lacked crucial visual design capabilities.

Collectively, these tools highlight a persistent challenge: a fundamental tension between visual accessibility (Composer, STEVE) and full expressive power (Lua2NCL, NCL Eclipse), often coupled with issues of scalability and poor integration into modern developer ecosystems. This landscape reveals a clear need for a new class of authoring system that can bridge this gap, offering both intuitive interaction and uncompromised technical capability. The present study takes a step in this direction, exploring approaches that move toward such integration.

3.2 NCL Validation as a Core Quality Component

Quality assurance is a critical dimension of NCL development, essential for managing the language's inherent complexity. Earlier contributions, such as the hybrid spatio-temporal validation approach proposed by (SANTOS et al., 2018), introduced the Simple Hypermedia Model (SHM) as a formal basis for checking consistency in declarative multimedia documents. By combining model-checking with Satisfiability Modulo Theories (SMT) solving, this work demonstrated that complementary techniques can be used to detect inconsistencies related to temporal and spatial relationships. Building on these foundations, more recent efforts are prominently represented by the NCL Validation Tool proposed by (COSTA et al., 2024), which evolved into the Extended Validation Framework. It was implemented as a Visual Studio

Code extension in TypeScript, this tool performs multi-layered validation, addressing syntactic, semantic, temporal, and resource-related constraints, and provides real-time diagnostics directly within a modern IDE. This represents a significant advancement over older, disconnected validators, confirming that tight integration into the development workflow is crucial for effectiveness.

However, even this modern approach remains fundamentally *reactive*. As described by its authors, the tool is designed to diagnose errors and present them to a human developer, who then performs the correction. Its purpose is to assist a human's authoring process, not to guide an automated one. In the context of automated content generation by an LLM-based agent, this reactive posture is insufficient. An intelligent agent cannot simply generate code and wait for an error message designed for human interpretation.

Instead, validation must evolve from a diagnostic tool into a *proactive, programmatic component*. It must become a co-pilot that an AI agent can consult and leverage **during** the generation process to ensure correctness by design. This conceptual shift from a human-centric error-checking mechanism to a machine-usable, generative quality assurance is a cornerstone of our research and a foundational requirement for building truly intelligent authoring systems.

3.3 AI-Powered Code Generation: Potential and Challenges

The emergence of Large Language Models has opened new frontiers for intelligent authoring. Systems like GitHub Copilot and AlphaCode excel at general-purpose programming, but their application to Domain-Specific Languages (DSLs) like NCL is not straightforward.

The foundational challenge of applying general-purpose LLMs to NCL was empirically confirmed in the seminal investigation by Moraes et al. (2023). Their systematic experiments, which tested models like GPT-3.5 and Google's Bard with prompts of increasing complexity, revealed profound limitations. For basic requests, the models often failed completely, either by "hallucinating" code in other programming languages or by incorrectly stating that NCL did not exist. More critically, even when provided with examples in a few-shot prompting approach, the generated code was structurally and semantically flawed. The study highlights specific failures, such as the generation of incorrect temporal logic in links and the use of undeclared connectors. These findings were pivotal, proving conclusively that simply applying standard prompt engineering techniques to general-purpose LLMs is insufficient for a specialized, rule-heavy domain like NCL. The work establishes a clear baseline: a smarter,

domain-aware architecture is not merely an improvement but a fundamental requirement for generating correct and reliable NCL code. Despite the limitations in output quality, this investigation stands out as the most impactful in terms of generating insights and providing conceptual inspiration. By systematically exposing the shortcomings of general-purpose large language models in the face of NCL's structural and semantic rigor, the study not only provided a diagnostic baseline but also illuminated critical design paths for future research.

To address such complexity, the field has moved towards multi-agent systems. While tools like **NCL Composer** struggle with scalability, frameworks like **Plan-and-Solve** (WANG et al., 2023) and **HuggingGPT** (SHEN et al., 2023) demonstrate how decomposing complex tasks among specialized agents can improve accuracy and handle greater complexity. More recent architectures like the **Self-Organized Multi-Agent Framework (SoA)** (ISHIBASHI; NISHIMURA, 2024) and **DyLAN** (LIU et al., 2024) introduce dynamic agent creation and selection, offering a path to build systems that can scale with task complexity, a direct answer to the scalability limits of earlier visual NCL editors.

Furthermore, advanced prompt engineering techniques extend the possibilities introduced by template-based tools like **NEXT**. While **Chain-of-Thought (CoT)** (WEI et al., 2022b) and **Tree-of-Thought (ToT)** (YAO et al., 2023) improve reasoning, it is **Retrieval-Augmented Generation (RAG)** (GAO et al., 2023) that holds the most promise. By enabling an LLM to access NCL specification documents and design patterns on-the-fly, RAG can support the creation of novel designs beyond predefined structures. This combination of multi-agent architecture and domain-specific knowledge retrieval forms the technical foundation for a truly intelligent authoring system.

3.4 Comparative Analysis and Research Gaps

This examination reveals several critical gaps that this dissertation aims to address:

1. **Mismatch Between General-Purpose AI and DSL Nuances:** As demonstrated by (MORAES et al., 2023), current AI systems struggle with NCL's specific constraints. Our research addresses this by designing a multi-agent architecture where agents are specialized for NCL's unique structural, temporal, and semantic requirements.

2. **Fragmented Development Workflow:** Tools like **NCL Composer** and **STEVE** are standalone applications, creating friction in modern workflows. By implementing our solution as a Visual Studio Code extension, we directly address this integration gap, embedding our tool within an ecosystem familiar to modern developers.

3. **Validation as an Afterthought:** Existing tools treat validation as a separate, post-processing step. Our approach implements its own validation mechanisms as a core, proactive component of the generative process, using them to guide the AI toward correct-by-construction outputs.

4. **Untapped Potential of Multi-Agent Systems for Hypermedia:** While frameworks like **Plan-and-Solve** are promising, their application to hypermedia authoring remains underexplored. We tailor a multi-agent collaboration strategy specifically for the challenges of NCL, such as separating temporal synchronization from spatial layout concerns.

5. **Lack of User Experience Evaluation:** Existing research has primarily emphasized technical metrics, leaving open questions about how advanced tools perform in realistic creative scenarios. Our work addresses this gap through a mixed-methods evaluation that examines both technical performance and practical usability.

3.5 Summary and Research Positioning

Table 3.1 highlights the limitations of existing approaches and explicitly aligns each with the corresponding contribution of this work.

Building directly on the gaps identified, this research introduces a domain-specialized, validation-integrated multi-agent architecture for NCL code generation, delivered through a visual interface embedded in a modern IDE. Rather than simply improving existing workflows, this approach aims to improve the authoring process, combining the precision of domain modeling with the adaptive potential of intelligent agents.

Approach	**Key Examples**	**Main Limitations**	**Gap Addressed by This Research**
Visual NCL Editors	NCL Composer, STEVE	Limited scalability; low integration with modern development workflows.	A scalable multi-agent backend and seamless integration as a VS Code extension.
Template-Based Tools	NEXT (XTemplate)	Oriented toward predefined templates, which streamline common design patterns but may not cover novel authoring needs.	Retrieval-Augmented Generation (RAG) to dynamically leverage documentation and examples for generating new designs beyond predefined structures.
Script-Based Generation	Lua2NCL	Requires programming expertise; lacks visual design capabilities.	Multimodal interfaces (visual, voice, text) to lower the barrier to entry without sacrificing power.
IDE Integration	NCL Eclipse	Platform dependency; purely textual approach excludes visual authoring.	A platform-agnostic approach within the dominant VS Code IDE, combining textual and visual authoring.
Validation Frameworks	Extended Validation Framework	Reactive rather than generative; operates separately from the authoring process.	Integration of validation as a proactive, core component of the AI generation workflow.
General LLM Code Generation	GitHub Copilot	Struggles with domain-specific constraints; inconsistent NCL structure adherence.	A specialized multi-agent architecture with domain-specific knowledge of NCL's semantics and structure.
Multi-Agent Systems	Plan-and-Solve, HuggingGPT	General-purpose focus; not adapted for hypermedia authoring requirements.	A bespoke multi-agent collaborative framework designed for hypermedia's unique challenges (e.g., time, space).

Table 3.1: Summary and Positioning of Related Work

4
Proposal: Tool for Automatic NCL Code Generation

This chapter presents the proposed integrated tool for automatic NCL code generation, designed to address the complexity barriers identified in traditional hypermedia authoring. The tool consists of two complementary components that emerged through an iterative development process: a visual interface for region base layout design and a multi-agent system for NCL code generation.

Building upon the theoretical foundations and limitations discussed in previous chapters, this research proposes a solution that combines visual authoring capabilities with AI-powered code generation to make NCL development more accessible and efficient for both novice and experienced developers.

4.1
Proposal Overview

The proposed tool addresses two core challenges in NCL authoring: the complexity of spatial layout specification and the need for a more accessible and intelligent workflow for building complete hypermedia applications. To this end, the solution integrates two complementary components within a unified Visual Studio Code extension:

1. **RegionBase Viewer**: A graphical interface designed to visualize and edit the spatial layout defined by the NCL code currently open in the workspace. This component provides real-time synchronization with the source code and allows authors to intuitively manage `<region>` elements, supporting better spatial organization and reducing the cognitive burden of dealing with nested layout structures. It also supports alternative interaction modalities, such as gesture and voice input, to enhance accessibility and creative flexibility.

2. **Conversational Code Generator**: A chat-based interface embedded within VS Code, inspired by tools like ChatGPT, Gemini, DeepSeek, and Claude, but tailored specifically for manipulating NCL code within the active workspace. Powered by a multi-agent architecture, this component enables users to issue natural language instructions to create, modify, or inspect elements of their NCL document, leveraging domain-specific knowledge to ensure the output is semantically correct and contextually coherent.

Together, these components enable an authoring workflow that blends direct manipulation with intelligent assistance, accommodating different user profiles and supporting both structured editing and exploratory creation. Figure 4.1 illustrates the overall architecture: on the left (1), the RegionBase Viewer; on the right (2), the conversational code generator powered by large language models.

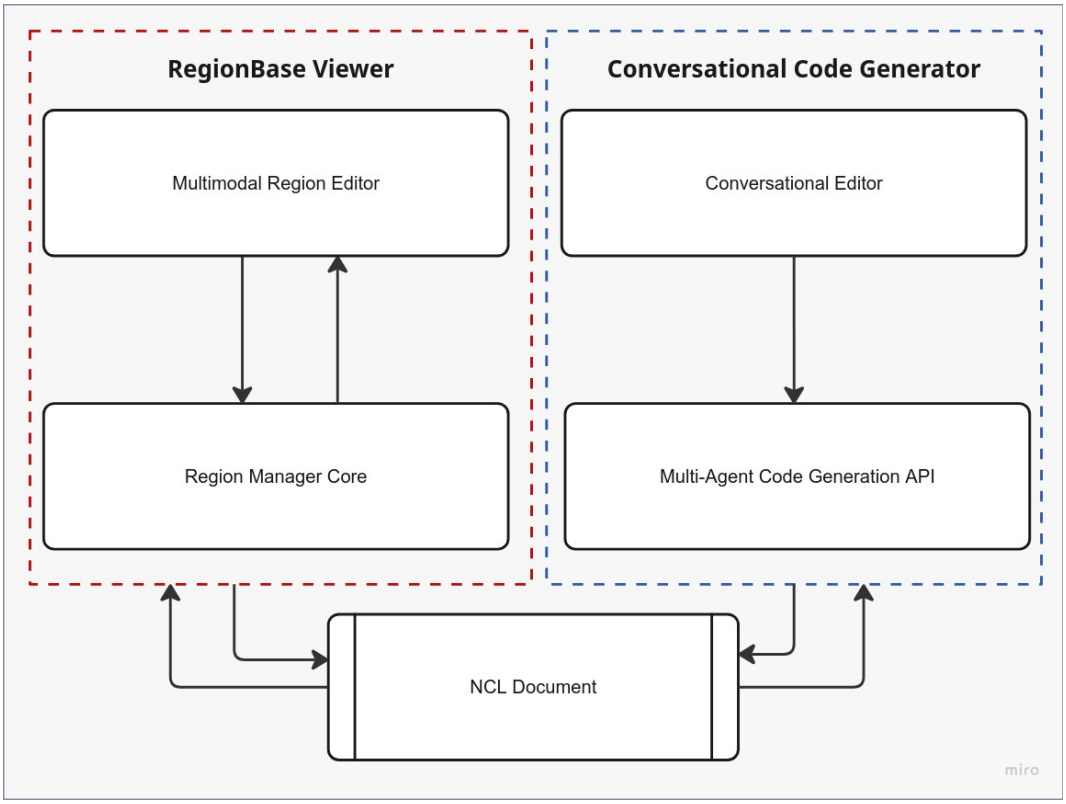

Figure 4.1: Conceptual model of the integrated authoring environment. The central NCL document serves as a reference, which can be modified through either a conversational AI workflow or a direct manipulation workflow using the visual interface.

The inner workings of the RegionBase Viewer and its interaction pipeline are discussed in detail in section 4.2. On the other hand, the conversational authoring interface builds upon a robust API for generating complete NCL documents using a Multi-Agent LLM architecture, as described in section 4.3.

4.2 Component 1: The RegionBase Viewer

The RegionBase Viewer corresponds to the visual authoring GUI depicted on the right side of Figure 4.2. It simplifies the spatial complexity of NCL by offering a canvas-based environment for viewing and interacting with the `regionBase` structure.

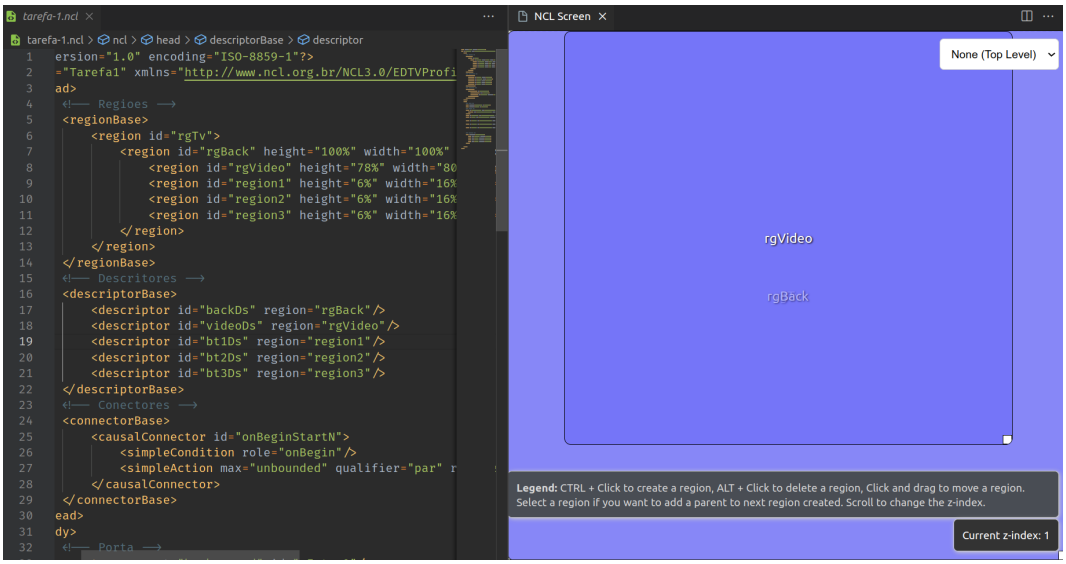

Figure 4.2: Visual region editor interface with drag-and-drop manipulation and real-time NCL code generation.

At the core of this component lies the **Multimodal Region Editor (MRE)**, the component responsible for capturing and handling all forms of user interaction within the Visual Studio Code WebView, whether through mouse, voice, or gesture. The MRE is primarily concerned with rendering the visual layout and dispatching user actions for processing. It delegates all logic and validation tasks to the **Region Manager Core (RMC)**, which acts as the authoritative layer for interpreting commands, enforcing the Region Manipulation Contract, and returning actionable updates, such as layout coordinates or operations to be applied to specific regions.

Implemented as a Visual Studio Code extension in TypeScript, the component parses the active NCL document to locate the `regionBase` node and its nested regions, extracting their spatial attributes (e.g., `id`, `width`, `height`, `top`, `left`). These elements are rendered as resizable and draggable blocks, providing a real-time visual representation of the document layout.

4.2.1 Region Manager Core

The **Region Manager Core (RMC)** is a centralized communication and coordination layer that mediates all interactions between the user interface and the NCL document structure. Its primary role is to enforce a *Region Manipulation Contract*, a standardized TypeScript interface that defines a consistent set of atomic operations—`createRegion`, `modifyRegion`, `deleteRegion`, `positionRegion`, and `resizeRegion`—used to manipulate region elements.

By routing all interaction requests through a standardized contract, the

RMC abstracts away the source of user input. Regardless of whether actions are triggered via mouse, voice, or gesture, they are treated uniformly as contract-compliant messages. This design enables a consistent processing pipeline, ensuring that the region layout remains semantically valid and synchronized with the NCL codebase.

Beyond interpreting commands, the RMC also manages states by receiving and emitting updates that reflect the spatial configuration of the layout. For instance, when a region is resized through the visual canvas or repositioned via a voice command, the system validates the operation, applies the change to the document model, and immediately reflects the update in the interface.

This contract-driven architecture promotes extensibility: new input modalities can be integrated without altering the underlying logic, as long as they conform to the expected message format. As a result, the tool supports both incremental editing and creation from scratch. An author may open an NCL file with a pre-existing `regionBase` and visually manipulate its structure, or begin with an empty canvas and construct a complete hierarchy interactively. In both workflows, the RMC guarantees that changes are consistently propagated to the underlying code.

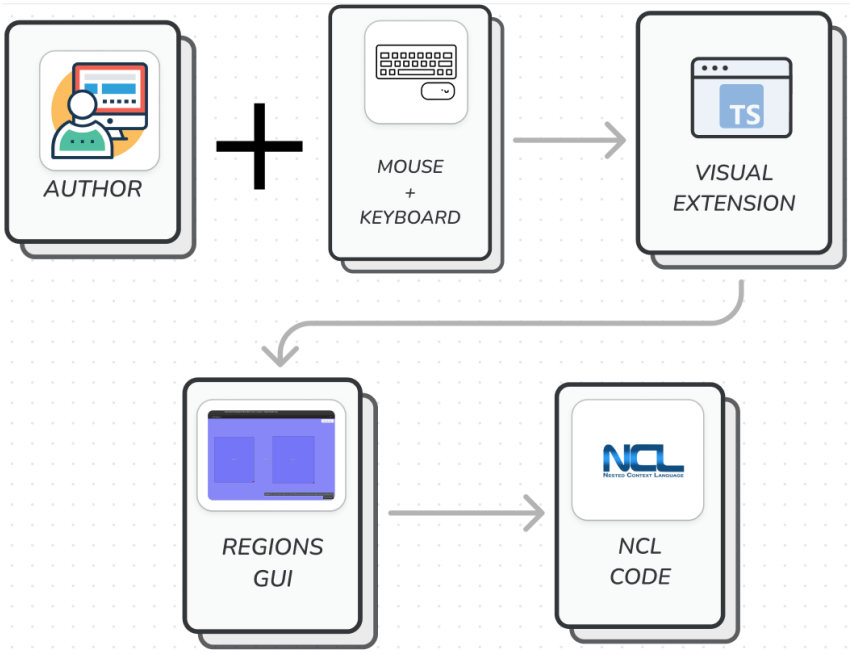

Figure 4.3: RegionBase Viewer Extension integrated into the VSCode IDE, supporting mouse and keyboard interactions for rendering images and generating NCL code.

Figure 4.3 illustrates this interaction flow, where traditional input devices (mouse and keyboard) are used to manipulate layout elements via

the graphical interface, with real-time synchronization between the view and the NCL document.

4.2.1.1 Extending RegionBase Viewer with Multimodal Inputs

Building upon its extensible architecture, the RegionBase Viewer goes beyond traditional mouse-based interaction by supporting the integration of alternative input modalities. This flexibility was leveraged in the development of a **Visual Multimodal Extension**, an optional add-on that enables users to interact with the interface using voice commands and hand gestures, without requiring changes to the system's core logic.

As illustrated in Figure 4.4, the user interface remains the same as in the baseline version of the tool Figure 4.2. The layout, functionality, and visual elements of the RegionBase Viewer are preserved; what changes is the input method. Instead of using a mouse to manipulate regions, users can issue spoken commands or perform gestures captured via webcam. This shift in modality is entirely transparent to the interface, thanks to the architectural separation between interaction logic and visual rendering.

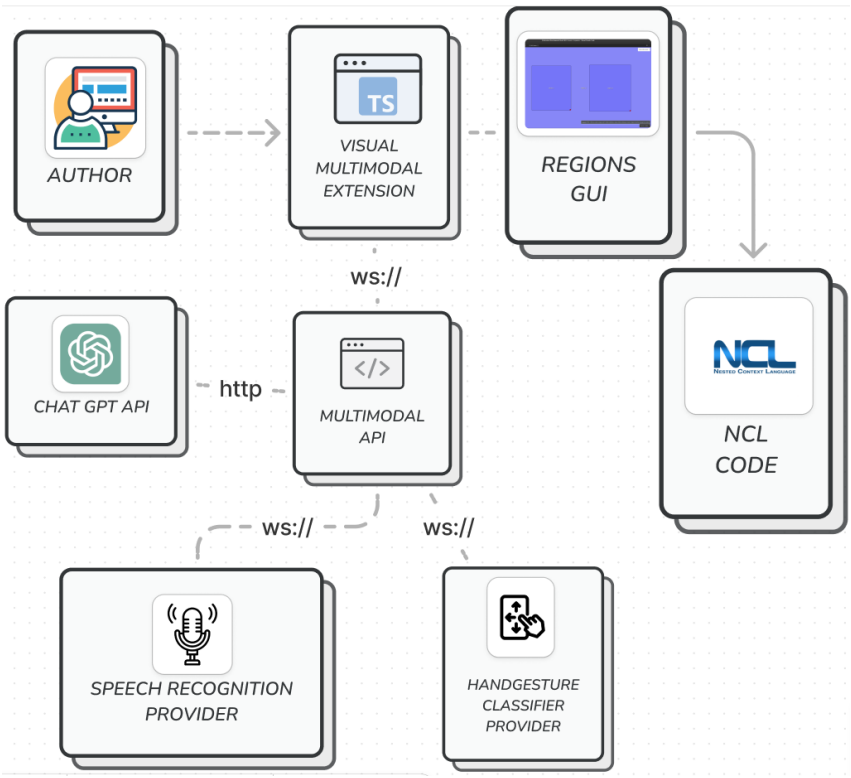

Figure 4.4: Visual Multimodal Extension integrating gesture and voice inputs through the Multimodal API.

To support this functionality, the extension communicates with a back-end service called the **Multimodal API (MMA)** via WebSockets. Acting as

a wrapper layer, the MMA translates inputs from diverse sources into contract-compliant commands defined by the RMC. It guarantees that all interactions, whether they originate from voice, gestures, or other modalities, are converted into structured messages with the expected format, ensuring consistent behavior across all input types.

The MMA in Figure 4.4 is composed of specialized providers that interpret specific types of input:

- **Gestures:** Using the MediaPipe framework (ZHANG et al., 2020), webcam-captured hand gestures are recognized and mapped directly to editing actions, e.g., pointing triggers a `positionRegion` command, while pinching initiates a resize.
- **Voice:** A three-stage pipeline handles voice-based commands: (1) speech is transcribed via Google Speech Recognition API (ZHANG, 2017), (2) the transcription is interpreted by a language model (GPT-3.5-turbo), which receives a structured prompt (see Code 1) and produces a JSON command, and (3) this command is sent to the extension as if the user had interacted directly with the canvas.

Code 1: Example prompt for LLM-based command interpretation.

```
1 You are an intelligent assistant that helps create and
      manipulate regions in a graphical interface.
2 The user's command is: "{command}"
3 Your job is to interpret the user's command and return the
      result as a JSON object following the structure defined
      below.
4
5 Example for creating a region:
6 {
7   "command": "createRegion",
8   "left": <value>,
9   "top": <value>,
10  "width": <value>,
11  "height": <value>,
12  "id": "<value>"
13 }
14 Now, interpret the user's command and return the result in
      this JSON format.
```

In practice, with this additional capability, this architecture enables a multimodal authoring workflow. An author can issue a voice command such as *"Add a region for the video in the lower right corner"*, and the system will insert the corresponding region into both the NCL document and the visual

canvas in real time. The author can then adjust their position or size using hand gestures, ensuring fluid coordination between modalities.

Importantly, the system respects user privacy: gesture detection via webcam is optional, requires explicit consent, and is processed locally on the user's machine. No visual data is stored or transmitted, and each frame is discarded immediately after processing.

This hands-free, flexible workflow showcases the architectural advantage of decoupling interaction mechanisms from authoring logic. By supporting new modalities through a contract-compliant interface, the system fosters innovation in hypermedia editing without sacrificing robustness, making it adaptable to other domain-specific authoring languages beyond NCL.

4.3 Component 2: Conversational Code Generator

Building on the successful implementation of the visual interface for editing the `<regionBase>` elements, an initial agent was designed to handle region-related generation tasks using a dedicated prompt structure. The results were notably accurate and consistent: the language model returned well-structured and semantically correct NCL code, confirming the effectiveness of delegating a well-scoped responsibility to a specialized agent. This success provided a strong indication that the hallucination issues identified in the work by Moraes et al. (2023), such as the invention of unsupported NCL elements or misuse of temporal constructs, could be mitigated by dividing the authoring task across focused agents, each operating within a narrow and controlled domain context.

Encouraged by these findings, the architecture was extended to cover the full set of NCL elements, including temporal synchronization (`<link>` and `<connector>` structures), media integration, interaction modeling, and conditional content adaptation. Each of these responsibilities is now handled by an individual agent, operating under the coordination of a high-level controller that orchestrates the generation process. This multi-agent strategy not only could improve the precision of each generated component but also enables richer and more coherent NCL applications to be authored via natural language input.

4.3.1 Conversational Editor

To make the tool accessible to end users, the architecture is exposed through a Visual Studio Code extension serving as the client interface. This

extension embeds a custom WebView component mimicking familiar conversational interfaces like ChatGPT, reducing the cognitive load associated with adopting new tools. The Conversational Editor GUI corresponds to the chat on the left side of Figure 4.5, illustrating the VS Code interface in action, showcasing how user input in natural language is processed by the system and transformed into structured NCL code. This example demonstrates the conversational user experience and the immediate feedback provided by the tool.

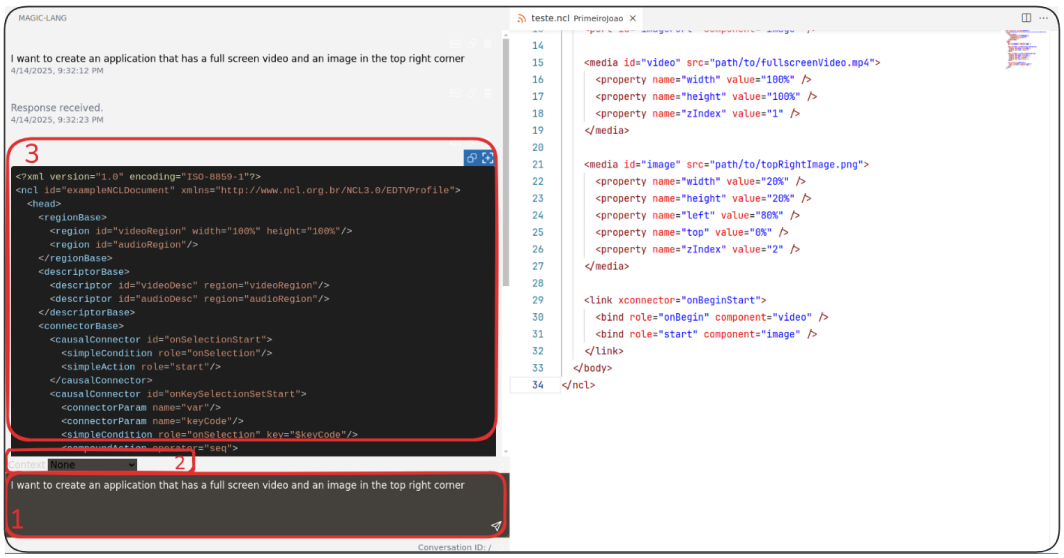

Figure 4.5: Interface of the agent-based authoring tool showing natural language input processing and automated NCL code generation

The interface provides two primary interaction paradigms:

- **Generate from Scratch**: Users provide open-ended natural language descriptions triggering the full multi-agent pipeline.
- **Correct Existing Code**: Users submit specific NCL code snippets for refinement, routing directly into validation and correction loops.

Both workflows pass through the same backend pipeline, with the extension acting as a lightweight frontend client sending HTTP requests to a centralized API that orchestrates all agent interactions. The use case logic for NCL document creation is exposed through RESTful endpoints, enabling seamless communication between the graphical interface and the multi-agent backend. API responses are rendered as syntax-highlighted XML, allowing users to copy or insert results directly into their workspace.

4.3.2 Multi-Agent Code Generation API

The goal of this architecture is to support developers in authoring complete NCL multimedia applications using natural language as the primary input. To ensure that the generated code is syntactically correct, semantically coherent, and free from hallucinations, the system adopts a multi-agent-based strategy. Each user query is decomposed into smaller, well-defined tasks that reflect distinct components of the NCL structure.

These subtasks are delegated to specialized agents, each responsible for handling a specific authoring dimension—such as region layout, temporal synchronization, media binding, or user interactivity. By distributing responsibilities across targeted agents and validating outputs at each stage, the system ensures that the final NCL output adheres strictly to domain rules.

Figure 4.6 presents an overview of this authoring pipeline, outlining the main phases: input interpretation, task decomposition, agent orchestration, code validation, and iterative refinement. This layered design promotes both reliability and flexibility, enabling the system to produce robust NCL documents even in complex and exploratory authoring scenarios.

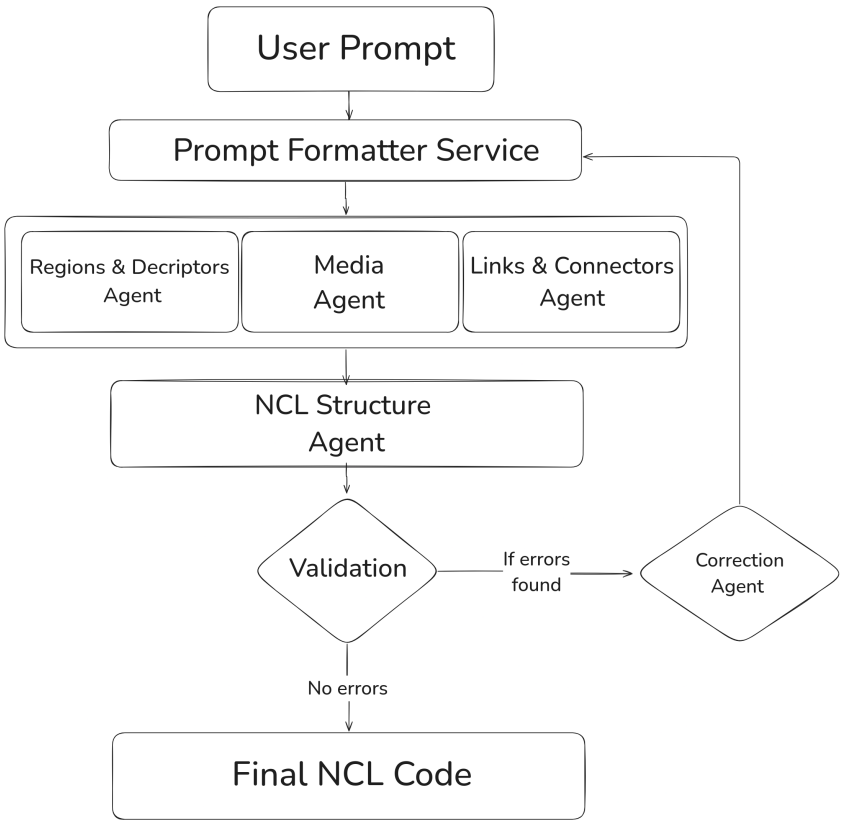

Figure 4.6: Diagram showing the architecture of the agent-based authoring tool for NCL documents

To operationalize the multi-agent tool, we developed an API responsible for orchestrating the generation of NCL documents through a modular and iterative pipeline. The implementation adheres to Domain-Driven Design principles, incorporating concepts such as *value objects*, immutable data structures that represent conceptual entities defined by their attributes rather than identity. Additionally, the system applies patterns from Hexagonal Architecture to enforce a strict separation between business logic and external interfaces, and adopts a layered design inspired by the principles of *Clean Architecture*, as proposed by Martin (2017), which emphasizes the encapsulation of business rules at the core, the inversion of dependency direction (from the outer layers inward), and the strict separation of concerns across well-defined boundaries; Figure 4.7 illustrates this architecture, where arrows point inward to denote the decreasing volatility and increasing stability of inner layers, ensuring that changes in external frameworks (e.g., switching from Express to Fastify) require minimal code refactoring, while all layers and their respective modules are covered by automated unit tests using the Jest framework, a strategy that guarantees each component is validated in isolation and enables confident refactoring and long-term maintenance.

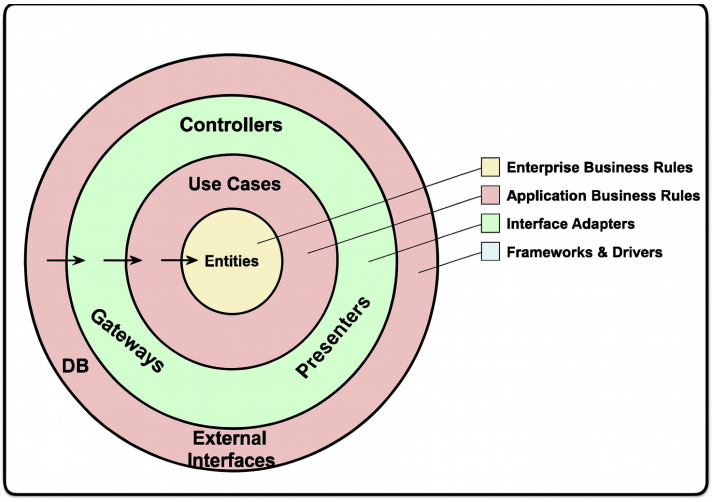

Figure 4.7: System architecture based on the Clean Architecture model (MARTIN, 2017). The inward-pointing arrows indicate decreasing volatility and stronger domain protection.

The summary presented below provides an overview of the four architectural layers that compose the system, highlighting their individual responsibilities and how they contribute to the overall modularity and maintainability of the platform. Each item outlines the primary role of a given layer, from the stable core of business rules to the more volatile outermost components responsible for interfacing with external systems. This high-level description

serves as a prelude to the detailed implementation analysis presented in subsection 4.3.3, where each layer is examined in depth with concrete examples from the codebase and emphasis on how the architectural principles are enforced in practice.

1. **Entities (Business Rules):** Represent the core domain logic and abstractions. These components remain independent of technical details and are the most stable elements of the architecture. In this implementation, the domain layer encapsulates all value objects, domain services, and business-specific contracts that model the semantics of NCL authoring. In this context, value objects are used to formally describe components such as `<region>`, `<media>`, `<descriptor>`, and others defined in the work of Costa et al. (2024). These objects encode the structural and semantic rules of each NCL tag and serve as the foundation for validating and composing NCL documents. They are consumed by the validation agent, which ensures that the generated code adheres to the correct syntax and semantics. Also at this layer, the `contracts` directory defines application-independent interfaces and domain-specific operations, such as agent coordination and validation logic, allowing high-level use cases to be built on stable abstractions.

2. **Controllers (Application Business Rules):** Encapsulate application-specific behavior, coordinating interactions between entities, and managing the execution of system operations. This layer defines the rules and workflows that drive how the system responds to external requests. In our implementation, the `application` layer includes use cases, services, and validation logic.

3. **Interface Adapters:** Adapt data between external systems and internal logic. This role is primarily fulfilled by the `infrastructure` layer, which contains gateways handling communication with ChatGPT, Together AI, and other providers, responsible for translating HTTP responses into domain-usable formats.

4. **Presentation Layer:** Represent the outermost layer, including tools and libraries like `Express`, `axios` and `bull-queue`, with adapters to handle contracts defined in the Business Rules Layer. These components are volatile and can be replaced with minimal impact on business logic, thanks to their strict isolation.

4.3.3 Implementation Details

In this section, we discuss the implementation decisions for each layer composing the Multi-Agent Code Generation API.

4.3.3.1 Presentation Layer (Frameworks and Drivers)

The presentation layer utilizes the Express framework to handle HTTP requests, exposing route adapters that translate incoming requests into standardized `HttpRequest` objects understood by the application layer. This layer is responsible for routing, serialization, and infrastructure-specific concerns, ensuring that external inputs are properly decoupled from core business logic.

To improve scalability and performance, the system employs a cluster-based strategy provided by Node.js. In this setup, the main process (master) forks multiple worker processes, each running an independent instance of the application. These workers share the same server port and handle incoming requests concurrently, allowing the system to utilize all available CPU cores fully. Each worker operates in isolation, and if one becomes unresponsive (e.g., due to a long-running task or memory leak), it is automatically terminated and replaced by a new one. This behavior enhances fault tolerance and ensures high availability by preventing system-wide failures caused by a single faulty process.

4.3.3.2 Application Layer (Controllers and Validation)

Controllers standardize request handling by performing input validation, delegating business logic, and formatting responses. They inherit from a base `Controller` class, which implements the Composite design pattern (RIEHLE, 1997) to compose validation logic. This base class defines a generic pipeline composed of three phases: validation, execution, and error handling. The `handle` method serves as the entry point for all controller requests, first invoking the `validate` method, which leverages a `ValidationComposite` to run multiple validation rules. These rules are constructed in the `buildValidators` method and may vary depending on the specific controller.

Concrete implementations, such as `CreateNclDocumentController`, override `buildValidators` to define request-specific validation logic and implement the `perform` method to invoke the corresponding use case. This architecture enforces a clean separation of concerns: validation is modular and

reusable, business logic is encapsulated in use cases, and response formatting is standardized.

The structure also provides robust error handling by mapping known exception types (e.g., `NotFoundError`, `ForbiddenError`, `AppError`) to consistent HTTP responses. This ensures predictable system behavior and simplifies debugging and maintenance across the application.

Code 2: NCL document creation controller with validation

```
export abstract class Controller {
  private validate(httpRequest: HttpRequest): Error | undefined {
    return new ValidationComposite(this.buildValidators(httpRequest)
        ).validate()
  }

  buildValidators(_httpRequest: HttpRequest): IValidator[] {
    return []
  }

  abstract perform(httpRequest: HttpRequest): Promise<HttpResponse>

  async handle(httpRequest: HttpRequest): Promise<HttpResponse> {
    try {
      const error = this.validate(httpRequest)
      if (error) return badRequest(error)
      const execute = await this.perform(httpRequest)
      return execute
    } catch (err) {
      const error = err as Error
      if (err instanceof NotFoundError) return notFound(error)
      if (err instanceof ForbiddenError) return forbidden(error)
      if (err instanceof AppError) return badRequest(error)
      return serverError(error)
    }
  }
}
export class CreateNclDocumentController extends Controller {
  constructor(private readonly createNclDocumentUseCase:
      ICreateNclDocumentUseCase) {
    super()
  }

  override buildValidators(httpRequest: HttpRequest): IValidator[] {
    const { prompt } = httpRequest.body as ICreateNclDocumentUseCase
        .Params
    return [...ValidationBuilder.of(prompt).required('prompt').build
        ()]
  }

  async perform(httpRequest: HttpRequest): Promise<HttpResponse> {
```

```
38      const output = await this.createNclDocumentUseCase.execute(
            httpRequest.body)
39      return ok(output)
40    }
41  }
```

4.3.3.3 Domain Layer (Entities, Use Cases and Services)

The core logic responsible for coordinating the multi-agent workflow resides in the domain layer, specifically within the `CreateNclDocumentUseCase` class. This use case serves as the main entry point for document generation, encapsulating the orchestration logic that integrates domain services and external agent invocations.

The `execute` method exemplifies the separation of concerns by delegating prompt decomposition to the `PromptFormatterService`, a domain service responsible for extracting structured sub-prompts (e.g., for regions, media, and links). These sub-prompts are then dispatched in parallel to specialized agents via the `AgentService`, which abstracts the communication with large language models. Once partial results are collected, a final agent is invoked to compose the full document structure. While the core orchestration resides in the `CreateNclDocumentUseCase` class, it is important to distinguish the conceptual roles of use cases and services in the architecture. A *use case* represents a concrete application-level operation that reflects a real user interaction or business scenario. It encapsulates the exact workflow needed to fulfill a specific goal, coordinating various components, such as domain services, value objects, and external agents, to deliver a meaningful outcome. Use cases are orchestrated by controllers and form the application layer's entry points.

In contrast, a *service* encapsulates a reusable and self-contained unit of domain logic that may be shared across multiple use cases. For instance, the `PromptFormatterService` and `AgentService` implement logic that is applicable in different scenarios involving prompt decomposition or LLM interaction, respectively. These services expose stable and composable behaviors that abstract underlying complexities without being tied to any single application flow. This separation ensures that the application layer remains thin and focused on orchestration, while domain logic is encapsulated in services with high cohesion and reusability.

The following code listing illustrates the implementation of the core use case responsible for orchestrating the multi-agent workflow. It integrates prompt decomposition, parallel agent invocation, document composition, and

validation into a single, cohesive execution flow that adheres to the separation of concerns principle.

Code 3: Core use case implementation coordinating multi-agent workflow

```
type Input = ICreateNclDocumentUseCase.Params
type Output = ICreateNclDocumentUseCase.Result

export class CreateNclDocumentUseCase implements ICreateNclDocumentUseCase {
  constructor(
    private readonly promptFormatterService: IPromptFormatterService,
    private readonly agentService: IAgentService,
    private readonly validatorService: IValidateDocumentService
  ) {}

  async execute(input: Input): Promise<Output> {
    const { links, media, regionsDescriptors } = await this.promptFormatterService.formatPrompt({
      prompt: new Prompt(input.prompt)
    })

    let generationPrompt = new Prompt(
      formatStructurePrompt(media.getValue(), regionsDescriptors.getValue(), links.getValue())
    )

    let result = ''
    let hasErrors = true

    while (hasErrors) {
      result = await this.agentService.callAgent({ prompt: generationPrompt })

      const evalAnswer = await this.agentService.callAgent({
        prompt: new Prompt(evaluationPrompt(result))
      })
      const needRetry = evalAnswer.trim().toUpperCase().startsWith('RETRY')

      if (!needRetry) {
        const syntaxErrors = await this.validatorService.validate({ document: result })
        if (!syntaxErrors || syntaxErrors.length === 0) {
          hasErrors = false
        } else {
          generationPrompt = new Prompt(correctionPrompt(syntaxErrors.join('\n'), result))
        }
      } else {
        generationPrompt = new Prompt(correctionPrompt(result))
```

```
40        }
41      }
42
43      return { nclDocument: result }
44    }
45 }
```

The execution flow demonstrates an iterative generation and validation process where:

1. The original prompt is decomposed into structured segments (regions, media, links)
2. Specialized agents are called in parallel to process each segment
3. A structure agent composes the final NCL document
4. A validation process ensures syntactic and semantic correctness
5. Iterative corrections are applied when necessary

To offer a deeper understanding of the responsibilities and inner workings of each agent involved in this pipeline, the following subsections provide a detailed analysis of the core domain services. Each agent encapsulates a distinct authoring dimension and operates within the domain layer, where responsibilities are clearly delineated to ensure modularity and testability.

4.3.3.4 Regions & Descriptors Agent

The **Regions & Descriptors Agent** is responsible for generating the `<regionBase>` and `<descriptorBase>` elements of an NCL document. It interprets spatial layout specifications and creates structured `<region>` elements with the appropriate attributes (such as `id`, `width`, `height`, and `zIndex`). For every identified region, the agent defines a corresponding `<descriptor>` that references the region and optionally includes an `explicitDur` attribute when temporal constraints are provided.

The prompt is designed to ensure consistency in structure and naming, encouraging the generation of region configurations and ensuring that each descriptor accurately maps to its respective region.

Code 4: Regions & Descriptors Agent prompt structure

```
1 You are an expert in NCL (Nested Context Language) regions and
    descriptors. Based on the following concepts, strictly follow
    the steps below to generate the correct NCL document.
```

```
2
3 Instructions:
4 1. Identify all regions mentioned in the command.
5 2. Determine the necessary properties for each region (id, width,
      height, zIndex).
6 3. Create a hierarchical structure to organize the regions with
      their specified properties.
7 4. Define the descriptors that reference each identified region. You
       may also include the ‘explicitDur‘ attribute to specify the
      explicit duration of media.
8
9 Example 1:
10 Command: "Create a region for a video at 80% width and 60% height,
      with a zIndex of 2."
11
12 <regionBase>
13   <region id="videoRegion" width="80%" height="60%" zIndex="2"/>
14 </regionBase>
15 <descriptorBase>
16   <descriptor id="videoDescriptor" region="videoRegion"/>
17 </descriptorBase>
18
19 Example 2:
20 Command: "Add a region for subtitles at the bottom of the screen,
      covering 100% width and 10% height."
21
22 <regionBase>
23   <region id="subtitleRegion" width="100%" height="10%" zIndex="1"/>
24 </regionBase>
25 <descriptorBase>
26   <descriptor id="subtitleDescriptor" region="subtitleRegion"/>
27 </descriptorBase>
28
29 Example 3:
30 Command: "Add a descriptor referencing the region frameReg with an
      explicit duration of 5 seconds."
31
32 <regionBase>
33   <region id="frameReg" width="50%" height="50%" zIndex="1"/>
34 </regionBase>
35 <descriptorBase>
36   <descriptor id="photoDesc" region="frameReg" explicitDur="5s"/>
37 </descriptorBase>
38
39 Your task:
40   - Return only the NCL document with the corrected definitions of
       regions and descriptors.
41   - Do not include explanations, comments, or any other type of
       additional text.
42
43 Command: <<prompt>>
```

This agent ensures that spatial arrangements and layout semantics are correctly formalized in NCL, laying the groundwork for consistent and structured multimedia presentations.

4.3.3.5 Media Agent

The **Media Agent** generates valid `<media>` elements along with associated `<area>` and `<property>` elements. When the user command includes a reference to a `descriptor`, the agent incorporates it directly into the generated element. However, this agent does not infer or define descriptors — it only retains references when explicitly mentioned.

The agent prompt is carefully structured with explicit instructions and curated examples, employing *Few-Shot Learning* to guide model generation. The prompt includes step-by-step rules that require each `<media>` element to contain `id` and `src`, and it enforces the proper placement of temporal attributes exclusively inside `<area>` tags.

Code 5: Media Agent prompt structure for NCL media element generation

```
You are an expert in defining <media> elements in NCL (Nested
    Context Language).
Based on the command below, strictly follow these steps to produce
    the NCL document:

Instructions:
1. Identify all <media> elements mentioned in the command.
2. Determine the required attributes for each <media>:
   - Each <media> must contain ‘id‘ and ‘src‘.
   - If the command references a descriptor, include ‘descriptor="
       descriptorName"‘ in the <media> element.
   - Do not create or infer descriptors on your own.
3. Identify any <area> elements:
   - Each <area> must contain ‘id‘ and ‘begin‘.
   - ‘end‘ is optional, but never place ‘begin‘ or ‘end‘ in <media>
       itself.
4. Include <property> elements if mentioned in the command, placing
    them within the corresponding <media>:
   - Each <property> must have ‘name‘ and ‘value‘.
   - Common properties include ‘width‘, ‘height‘, ‘left‘, ‘top‘, ‘
       zIndex‘, and ‘explicitDur‘.
   - Correct any typos such as "heigth" to "height".
5. Ensure that the ‘begin‘ and ‘end‘ attributes (in <area>) are
    always in seconds (e.g., "12s", "41s").
6. Generate only the NCL document containing:
   - The <media> elements (with attributes and children).
   - The <area> elements (if mentioned).
   - The <property> elements (if mentioned).
```

```
7. Do not include any explanations, comments, or extra text.
8. Never place ‘begin‘ or ‘end‘ inside <media>.

Command: <<prompt>>
```

By constraining the output format and avoiding explanatory content, the Media Agent ensures that the resulting NCL fragments are ready to be parsed and integrated into the broader authoring pipeline.

4.3.3.6 Links & Connectors Agent

The **Links & Connectors Agent** interprets user intent related to media synchronization and interaction, generating appropriate NCL `<link>` elements with corresponding causal connectors. This addresses one of NCL authoring's most challenging aspects, governing document interactivity and temporal behavior through often verbose and rigid syntax.

The agent prompt leverages *Retrieval-Augmented Generation (RAG)* and *Few-Shot Learning* strategies. In this setting, retrieval is performed implicitly by embedding a handcrafted memory, containing all relevant interaction patterns, directly within the prompt, enabling the agent to reason over 96 predefined NCL connectors without external lookup calls. A key element of this memory is the `ConnectorsEnum`, which maps high-level semantic roles to their corresponding NCL syntax definitions, supporting accurate and context-aware code generation within constrained prompt windows.

Code 6: Links & Connectors Agent prompt for NCL interaction generation

```
You are an expert in NCL (Nested Context Language) link and
    connector formatting.
Your task is to analyze the following command and return the most
    appropriate
connector(s) from the provided enum.

Instructions:
1. Review the command carefully.
2. Generate at least three candidate answers using different
    reasoning paths,
   considering edge cases and interpretation possibilities.
3. Select the most consistent and contextually accurate connector(s)
     based on
   candidate answer comparison (self-consistency strategy).
4. Return the NCL code string(s) corresponding to selected connector
    (s) from
   the provided list, without additional formatting.

Available Connectors Enum:
```

```
<<Object.entries(connectorsEnum)
  .map(([key, value]) => `<<key>>: <<value>>`)
  .join(', ')>>

Response Format:
- Return final selected NCL code string(s) that best fit the command
- Return only the code string, without backticks or extra symbols
- Do not include explanations, comments, or additional text

Command: <<prompt>>
```

This structured, minimal output format ensures compatibility with downstream parsers while avoiding ambiguities. By isolating connector selection from connector expansion, the system supports model-agnostic reasoning and improves overall results.

4.3.3.7 NCL Structure Agent

The **NCL Structure Agent** serves as the final synthesis point, assembling complete, cohesive, and standards-compliant NCL documents. Unlike earlier agents handling isolated tasks, this agent reasons about the entire document structure, ensuring all components are integrated adequately within NCL syntax and semantic rules.

The agent prompt employs the *Tree-of-Thoughts* reasoning strategy, encouraging the exploration of multiple potential structural compositions, evaluating coherence, and selecting the most consistent representation. This is particularly valuable for NCL's declarative nature with hierarchical nesting and strict ordering between elements.

Code 7: NCL Structure Agent prompt for document assembly and finalization

```
You are an expert in NCL (Nested Context Language) document
    generation.
Your task is to generate the full structure of the NCL document in
    XML format,
integrating the provided media, descriptors, and links results. Use
    a Tree of
Thoughts approach to ensure all components are properly organized.

Instructions:
1. Infer and include necessary ports based on media and links
    provided,
   ensuring components requiring external access are properly linked
       through ports.
2. Integrate main components of the NCL document, including header,
    body,
```

```
10    and all required sections.
11 3. Relate each part of the document (descriptors, media, links) so
      all
12    interactions and hierarchies are clear and correctly established.
13 4. Generate the complete NCL document by combining results from
      media,
14    descriptors, and links as required by the command.
15
16 Response Format:
17 - Return only the complete NCL document
18 - Do not include explanations, comments, or additional text
19
20 Expected Structure: [XML hierarchy example]
21 Command: Use provided components to construct full document,
      inferring ports when applicable.
```

The agent infers additional structural elements not explicitly provided, such as document header metadata, connector base definitions, port declarations, and document identifiers, which are needed for semantic completion. It resolves implicit references and cross-references to relevant components, injecting missing structures as needed.

4.3.3.8 Correction Agent

The **Correction Agent** operates within the iterative refinement loop, receiving NCL documents flagged as invalid and applying intelligent corrections based on detailed diagnostic feedback. This ensures that generated outputs conform to NCL standard requirements and meet structural consistency expectations.

Unlike generative agents that start from broad user intent, the Correction Agent operates reactively, taking as input the current NCL document version and diagnostic reports that list validation errors with precise line and column references.

Code 8: Correction Agent prompt for document refinement

```
1 You are an expert in NCL (Nested Context Language) document
      validation and correction.
2
3 Your task is to critically review and correct the following NCL
      document
4 using a self-consistency strategy, where multiple reasoning paths
      are explored
5 before selecting the most accurate solution.
6
7 Instructions:
```

```
1. Analyze the document and identify syntactic, structural, or semantic errors
   based on the validation context provided below.
2. Generate three different corrected versions of the document, using distinct
   reasoning paths (e.g., different interpretations or resolution strategies).
3. Compare the three versions and select the one that best aligns with NCL
   standards and document context.
4. Return only the final selected version.

Validation Context:
<<validateCode>>

Goal:
Produce a corrected version of the NCL document that is valid, coherent, and well-structured.

Your response must include only the corrected NCL document,
without explanations, comments, or formatting.

Current document to analyze:
<<result>>
```

The prompt employs *Self-Consistency* strategies, prompting iterative solution refinement and convergence toward consistent correction. The agent makes targeted adjustments rather than regenerating entire documents, preserving previously validated sections.

The Correction Agent supports recursive invocations, with new documents re-validated after each correction cycle. If issues persist, the process repeats with updated diagnostic reports until documents pass validation or maximum correction attempts are reached, introducing resilience and fault tolerance into the authoring workflow.

4.3.3.9 Infrastructure Layer (Adapters and Gateways)

External integrations follow dependency inversion principles. Gateways like `AgentGateway` abstract the complexity of communicating with LLM providers (e.g., OpenAI, Together.AI, Claude, DeepSeek), handling authentication and formatting while isolating external changes from the domain layer.

This infrastructure layer ensures that:

- Changes in external APIs do not affect business logic
- Different LLM providers can be easily interchanged

– Authentication and formatting are handled consistently
– The system is resilient to external communication failures

The gateway pattern implementation maintains a clean separation between domain concerns and external service communication, allowing for flexible provider switching and robust error handling strategies.

4.4 Considerations

With the design and implementation of the tools established, it becomes essential to evaluate how NCL code authors perceive their usability, particularly in terms of interaction flow, learning curve, and overall effectiveness. This evaluation aims to understand the extent to which the proposed system supports real authoring scenarios and aligns with the needs of its target users. The details of this assessment are presented in chapter 5.

5
Experiments and Evaluation

This chapter presents the evaluation of the proposed NCL authoring tool, examining both the RegionBase Viewer and the Conversational Code Generator components through distinct but complementary experimental approaches. The review offers insight into the practical utility, usability characteristics, and potential impact on NCL development workflows, thereby enhancing our understanding of how AI-driven and visual tools can transform domain-specific language authoring.

5.1
Ethical Considerations and Research Integrity

All phases of this research followed ethical guidelines for studies involving human participants. Volunteers signed informed consent forms (Appendix A) and were fully informed about their rights and the study's purpose. No personal data was captured and stored, and all experiment data collected was anonymized and handled securely. The research protocol was approved by the Ethics Committee of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), under Decision No. 134/2024, Protocol 111-2024.

5.2
RegionBase Viewer Evaluation

To assess the effectiveness and usability of the RegionBase Viewer, we conducted a user study comparing three distinct interaction modalities:

- **Baseline Scenario**: Traditional code editing using only the VS Code text editor without visual assistance;
- **Mouse-keyboard Interface Scenario**: Using the RegionBase Viewer with conventional mouse and keyboard interaction;
- **Multimodal Interface Scenario**: Using the RegionBase Viewer Tool with voice commands and gesture recognition capabilities.

The evaluation strategy was designed to examine both quantitative performance metrics and qualitative user experience factors.

5.2.1 Experiment Design and Methodology

The evaluation employed a within-subjects experimental design to minimize individual differences and allow direct comparison of interaction methods. It combined quantitative metrics from collected data and surveys with qualitative insights from post-experiment survey interviews to assess the tool's effectiveness.

For all the scenarios previously described, the experiment consisted of the completion of three progressive tasks involving the creation and manipulation of regions within NCL documents:

1. **Region Creation**: Creating 5 new regions with specific dimensions and positions, following the layout illustrated in Figure 5.1
2. **Region Resizing**: Modifying previously created regions to match a target layout, as shown in Figure 5.2
3. **Region Repositioning**: Moving and resizing regions to achieve different spatial arrangements, demonstrated in Figure 5.3

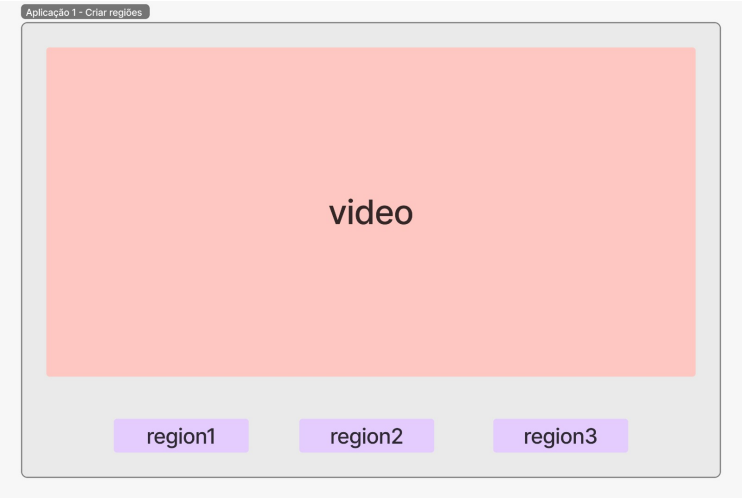

Figure 5.1: Task 1: Create 5 regions reproducing the defined layout

These tasks were specifically designed to evaluate the core functionality of the RegionBase Viewer while progressively increasing complexity to assess different aspects of the interface's usability and effectiveness.

To minimize learning effects and order bias, all sessions began with the **Baseline Scenario**, while the order of the **Mouse-keyboard Interface Scenario** and the **Multimodal Interface Scenario** was randomized across participants. This design allowed for a direct comparison between traditional and enhanced authoring methods.

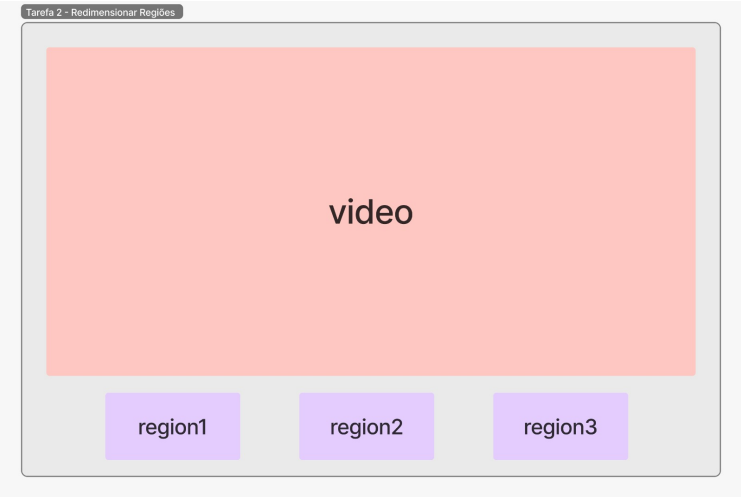

Figure 5.2: Task 2: Resize existing regions to match target layout

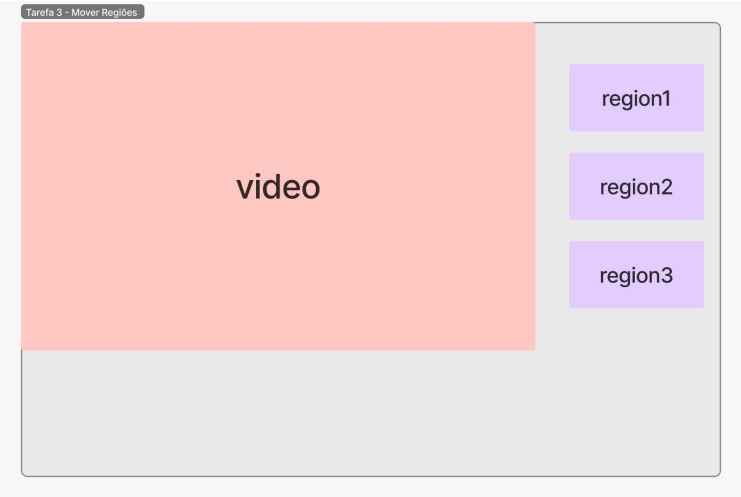

Figure 5.3: Task 3: Move and resize regions to achieve new spatial arrangement

For each task, participants were instructed to monitor **Completion time** and submit the resulting NCL documents in each scenario.

Following each experimental scenario, participants also completed a questionnaire inspired by the System Usability Scale (SUS) (BANGOR; KORTUM; MILLER, 2008; LEWIS, 2018). The questionnaire included questions related to usability and satisfaction perception, tailored to the context of region manipulation.

1. **Ease of Use**: Intuitiveness of the interface and interaction methods
2. **Efficiency**: Perceived speed of task completion
3. **Accuracy**: Evaluation of the correctness of created regions compared to target layouts
4. **User Satisfaction**: Overall satisfaction with the authoring experience

Participants rated their agreement with four statements mapping to the usability dimensions using a 7-point Likert scale (JOSHI et al., 2015) (1 = Strongly Disagree, 7 = Strongly Agree):

No	**Statement**	Dimension
S1	I think it is easy to use the tool to create and edit regions	Ease of Use
S2	I was able to create and edit regions using the tool	Efficiency
S3	I think the regions created and edited with the tool are correct	Accuracy
S4	I am satisfied with using the tool to create and edit regions	User Satisfaction

Table 5.1: Usability statements of the questionnaire used in the RegionBase Viewer experiment

After the completion of all three scenarios, we also asked participants to answer a post-experiment question to assess the participants' **Preference** among the interaction modalities.

The experiment was conducted individually and remotely to ensure participant comfort and minimize external factors. Each session followed a standardized protocol:

1. **Preparation Phase**: Participants received detailed installation instructions and usage scripts for the VS Code extension
2. **Task Briefing**: Clear explanation of each task with visual examples and expected outcomes
3. **Material Provision**: Distribution of incomplete NCL documents and all necessary assets for task completion
4. **Task Execution**: Supervised completion of tasks under each experimental condition
5. **Data Collection**: Immediate post-condition questionnaire completion to capture fresh impressions
6. **Debriefing**: Optional follow-up discussion for deeper insights and improvement suggestions

Before the experiment, a one-hour pilot experiment was conducted to validate the experimental design, assess question clarity, and refine instructional materials. Based on pilot feedback, corrections and improvements were implemented to ensure the validity and reliability of the evaluation process.

5.2.2 Participant Selection and Characteristics

The experiment was conducted remotely with 11 participants individually. Each participant was selected based on their familiarity with multimedia content editing and programming experience in Visual Studio Code.

To understand how prior experience influenced performance and satisfaction, we collected demographic information, including:

- Programming experience level and background
- Familiarity with Visual Studio Code
- Previous exposure to NCL or multimedia authoring tools
- Experience with multimodal interfaces (voice commands, gesture recognition)

This information enabled a correlation analysis between participant background and interface preferences, providing insights into the tool's accessibility for users with varying levels of expertise.

5.2.3 Results and Analysis

The experimental evaluation compared three authoring modalities: text editor, traditional mouse-keyboard interactions (hereafter referred to as **MKI**), and multimodal interactions combining voice and gesture (**VGI**) based on participant performance and perception across usability metrics.

Each participant engaged with all interfaces, allowing within-subject comparisons and enabling a richer understanding of the trade-offs involved in each interaction style.

Table 5.2 summarizes the overall results collected through post-task questionnaires and performance metrics. Only 7 of the 11 participants completed all experimental phases and provided complete data for analysis. Each session lasted approximately 50 minutes, during which participants completed standardized region manipulation tasks under all three experimental conditions.

Metric	Text Editor	MKI	VGI (Multimodal Extension)
Ease of Use (median)	5.0	7.0	5.0
Accuracy (median)	6.0	7.0	6.0
Overall Satisfaction (median)	5.0	6.0	5.0
User Preference (%)	0%	71%	29%
Task Completion Time (avg)	12.3 min	8.7 min	14.1 min

Table 5.2: Comparative Performance Summary Across Interaction Modalities: Text Editor, MKI, and VGI

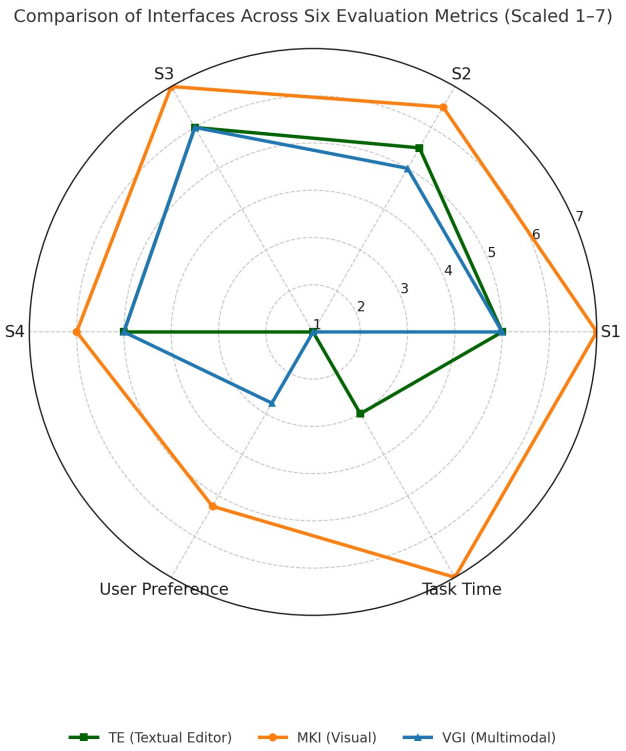

Figure 5.4: Radar chart comparing normalized evaluation metrics (scaled 1–7) across three interfaces: Textual Editor (TE), Visual Graphical Interface (MKI), and Multimodal Gesture-based Interface (VGI). The chart includes four median. Higher values indicate more positive evaluations or better performance. MKI outperforms both TE and VGI in all dimensions, while VGI shows slightly lower ratings than TE, particularly in Task Completion Time.

The MKI modality demonstrated superior performance across all measured dimensions, achieving the highest median ratings for ease of use (7.0 vs. 5.0 for alternatives) and accuracy (7.0 vs. 6.0). Given the exploratory nature of this study and the small sample size ($n = 7$), we focus on descriptive analysis and qualitative insights rather than inferential statistical testing.

These findings suggest that the MKI modality, which integrates traditional mouse-based interaction with visual feedback, offers a favorable balance between intuitiveness and precision, as seen in Figure 5.4. Participants consistently reported higher ease of use and satisfaction when using this interface, likely due to the familiarity of point-and-click paradigms and the immediate visual representation of editing actions.

Although the VGI modality showed potential, particularly in enabling hands-free interaction, its higher task completion time and lower preference ratings may reflect current limitations in gesture and voice recognition reliability. Notably, no participant preferred the text editor, underscoring the importance of visual affordances and guided workflows in lowering the barrier to authoring in NCL.

5.3 Conversational Code Generator Evaluation

To assess the effectiveness and usability of the multi-agent NCL code generation system, we conducted a qualitative study focusing on human-centered evaluation of the tool's practical utility, interaction quality, and workflow integration. Understanding that the system represented a more complex solution compared to the *RegionBase Viewer*, we opted for a methodological approach capable of capturing nuanced aspects of user interaction and perception.

The evaluation strategy employed an observational study design with post-task interviews to collect both behavioral patterns and subjective experiences. Given the specialized nature of NCL development and the limited pool of experienced practitioners, we adopted a qualitative research approach emphasizing depth over breadth. This methodology enables a detailed exploration of user behavior, interaction patterns, and emergent needs that might not surface through large-scale quantitative studies. The evaluation framework was designed to examine three critical dimensions:

1. **Technical Effectiveness**: Assessment of the system's ability to generate syntactically correct and semantically coherent NCL documents from natural language descriptions across varying complexity levels
2. **User Experience Quality**: Evaluation of interaction intuitiveness, learning curve, and overall perception of satisfaction with the conversational authoring paradigm
3. **Workflow Integration**: Analysis of how the tool fits into existing development practices and its potential for transforming NCL authoring workflows

5.3.1 Experiment Design and Methodology

The study followed a structured four-phase protocol designed to evaluate the tool's capabilities and user experience comprehensively:

– **Phase 1: Introduction and Contextualization**
Each session began with a standardized introduction covering the study objectives, tool overview, and installation demonstration. Participants received a brief explanation of the tool's functionality and were provided with the complete set of media assets (videos and images) to be used

throughout the tasks. The initial assessment captured participants' experience with NCL, code generation tools, LLMs, and VS Code extensions to establish baseline familiarity levels.

– **Phase 2: Task Execution and Observation**
Participants completed NCL document creation tasks at two complexity levels to evaluate how the multi-agent system handles varying difficulty requirements:

Simple Tasks: (i) Create an NCL document with a video and an overlaid logo image representing a broadcaster; (ii) Create an NCL document with two videos playing sequentially, where the second starts after the first ends.

Complex Task: Create an NCL document with two full-screen videos where the second video starts automatically after the first ends, and include an interactive button that allows switching between videos at any time.

All tasks involved code generation from scratch using natural language commands, requiring the system to interpret requirements for multiple elements, synchronization, and interactivity without prior code foundation.

– **Phase 3: Observational Data Collection**
During task execution, data collection included task completion times, observable user behaviors, decision-making patterns, interaction hesitations, and spontaneous verbal feedback through think-aloud protocols. When authorized, screen recordings, agent command logs, and generated NCL code versions were also captured for technical analysis. A structured observation form was used to document real-time behavioral patterns and participant comments systematically.

– **Phase 4: Post-Task Interview**
Semi-structured interviews explored participant perceptions across eight key dimensions: (1) the installation process and setup experience; (2) visibility and accessibility of the tool within the VS Code interface; (3) intuitiveness of commands and identification of interaction challenges; (4) ease of initial adoption and learning curve; (5) perceived utility and suggestions for additional features; (6) workflow impact and friction points; (7) unexpected or confusing behaviors; and (8) perceived quality and usefulness of the generated NCL code, including preferences and necessary manual edits.

5.3.2 Participant Selection and Characteristics

Seven participants with relevant technical backgrounds participated. The group consisted of graduate students in STEM fields, software engineers, and academic researchers. All participants possessed undergraduate-level programming knowledge and prior experience with both VSCode and generative AI tools (e.g., ChatGPT, Gemini).

- **Technical Background**: All participants possessed at least undergraduate-level programming knowledge and familiarity with software development concepts
- **AI and Authoring Experience**: Each participant had prior experience with code generation tools, authoring tools, or Large Language Models (GPT, DeepSeek, or similar systems)
- **NCL Knowledge**: Participants varied in their NCL familiarity, ranging from academic exposure in multimedia systems courses to professional experience in digital TV projects
- **Development Environment Familiarity**: Consistent experience with Visual Studio Code and extensions as a primary development environment, ensuring focus on tool evaluation rather than editor navigation

This participant profile provided the technical baseline necessary for advanced system engagement while reducing onboarding requirements and enabling contextualized feedback during task execution.

5.3.3 Results and Analysis

To assess the effectiveness of the proposed multi-agent system, we adopted a mixed-methods approach that combined quantitative performance metrics with qualitative user insights.

The quantitative analysis focused on:

- Measuring and comparing average task completion times across complexity levels
- Identifying recurring error types and their frequencies
- Benchmarking against traditional NCL development workflows

All participant behaviors, utterances, and post-task feedback were systematically documented and analyzed through iterative cycles of coding and

thematic grouping. This process continued until theoretical saturation was reached—i.e., when no new themes emerged.

As a result of this analysis, nine core themes were identified based on user interactions and feedback. Table 5.3 presents the final synthesis of findings, including representative quotes that illustrate typical perceptions. While exact phrasing varied, the selected quotes are conceptually aligned with the broader themes observed across participants.

Thematic Category	**Representative Quote (Translated from PT)**	**Participants**
1. Model interpretation and accuracy	"I asked to place the logo in the upper corner, but it put it closer to the center."	P1, P3, P4
2. Usability and interaction feedback	"It would be nice to show the video result immediately."	P2, P5
3. Onboarding and user experience	"I didn't even need to read a tutorial; it felt like a normal chat."	P2, P3, P5
4. Workflow efficiency and perceived value	"I did in 5 minutes what used to take 30."	P1, P4
5. Code clarity and validation effort	"I had to look line by line to find the error."	P3, P4, P7
6. Prompt handling and context limitations	"It didn't understand more generic commands."	P2, P5
7. Suggestions for improvement	"It would be better to edit directly in the chat and retain the context between messages."	P2, P5

Table 5.3: Synthesis of findings base on thematic grouping of participant interviews and observations.

The most prominent finding was the substantial productivity gain enabled by the system. Participants reported completing authoring tasks in as little as 5 minutes, compared to 30 minutes using conventional tools. This time reduction was especially pronounced among users with intermediate NCL experience, who benefited most from the system's abstraction of structural and syntactic complexity.

Efficiency gains, however, varied depending on user expertise. While novice users appreciated the simplification of the authoring process and found it educational, expert users occasionally found the conversational interface slower for simple edits, preferring direct code manipulation in such cases.

A recurring limitation involved spatial interpretation. Four participants observed discrepancies between their intended layout descriptions and the system's generated positioning. For example, requests to place an element in the upper corner occasionally resulted in center-aligned outputs. These findings

highlight the challenges of translating natural language spatial cues into precise coordinate definitions in NCL.

Advanced users also pointed out the system's limitations for pixel-perfect adjustments, which are often critical in broadcast production. While the multi-agent system handled structural layout competently, its ability to perform fine-grained spatial tuning remains limited, indicating the need for enhanced spatial reasoning or hybrid manual support. Additionally, some expert participants noted that the system occasionally produced unnecessarily verbose or suboptimal NCL structures. In such cases, alternative logical formulations have led to more efficient and readable code, suggesting opportunities for optimization in the generation pipeline.

From a usability perspective, the conversational interface significantly lowered the entry barrier. Participants described the interaction as intuitive and familiar, likening it to explaining their goals to a knowledgeable assistant rather than wrestling with syntax. Notably, this led to unexpected educational value: several participants mentioned that observing how the system structured the generated code helped them understand NCL more clearly than reading documentation alone. This suggests the system's potential as both a productivity tool and an onboarding aid for newcomers to NCL.

Nonetheless, participants emphasized the importance of code validation. Despite positive impressions, users reported reviewing each generated line to ensure correctness, underscoring the principle of "trust but verify" in AI-assisted development. Some also experienced difficulty recovering from errors—once incorrect code was produced, continuing the conversation without losing context was sometimes challenging.

Backend logs confirmed effective coordination among specialized agents. The Structure Agent successfully integrated content from Media, Regions, Descriptors, and Connectors agents to assemble valid NCL documents. On average, **1.8** validation iterations were required for code convergence, with most issues being syntactic rather than semantic. Together, these insights reveal that while the system offers notable productivity and educational benefits, further improvements are needed in error recovery, spatial reasoning, and iterative refinement to enhance usability and trust.

5.4 Considerations

The evaluation of both components, the RegionBase Viewer and the Conversational Code Generator, revealed complementary strengths and limitations that offer valuable insights for the future of domain-specific language tooling.

Across both evaluations, several recurring themes emerged. First, the importance of immediate visual feedback was consistently emphasized. The RegionBase Viewer was praised for its intuitive representation, while participants frequently requested real-time previews in the Conversational Code Generator, highlighting a shared user need in multimedia authoring.

Second, learning curve dynamics differed in meaningful ways. Traditional interaction paradigms, such as the RegionBase Viewer, offered immediate usability with minimal onboarding. Interestingly, the multi-agent conversational interface also demonstrated a low entry barrier, with participants reporting that it felt intuitive and aligned with familiar chatbot interactions. Beyond ease of use, the system provided educational benefits, especially for NCL novices, who reported gaining structural insights simply by observing the generated code. Both tools reinforced the notion that AI assistance is most effective when designed for augmentation rather than automation. Users consistently verified and corrected outputs, underscoring the need for transparency, user control, and validation mechanisms in human-AI collaborative systems.

Tool effectiveness also varied by task. Visual tools excelled in scenarios requiring spatial accuracy, conversational agents supported rapid prototyping and onboarding, and text editors remained preferred for precise manual control. These findings suggest that multi-modal or hybrid environments may offer the best support across diverse authoring needs.

6
Conclusions

This dissertation addressed the challenge of authoring interactive applications using the Nested Context Language. In response to the steep learning curve and limited tooling traditionally associated with NCL, we proposed, implemented, and evaluated a authoring system composed of two integrated components: a multi-agent system for conversational code generation and a multimodal visual interface for spatial editing and live preview. Together, these components, packaged as a Visual Studio Code extension, aim to streamline the authoring process and broaden access to NCL-based development.

The research was guided by the goal of designing a software architecture that integrates conversational AI and visual editing components in a synchronized authoring workflow. To that end, we implemented a multi-agent API capable of interpreting natural language commands and generating valid NCL code, while also developing a visual interface for creating and editing NCL elements through direct manipulation. These efforts were accompanied by an robust evaluation conducted with developers to assess usability and understand the tool's impact on the authoring workflow.

The primary contribution of this work lies in its architectural integration of intelligent language models and direct manipulation interfaces, establishing a unified workflow in which users can fluidly transition between conversational and visual authoring. The multi-agent system interprets natural language prompts and generates semantically coherent NCL code by orchestrating a series of specialized agents, each responsible for distinct structural aspects of the language. Meanwhile, the visual interface enables region editing through mouse, voice, and gesture interactions, anchored in a contract-based structure that ensures consistency across modalities.

A user study combining quantitative and qualitative methods demonstrated the system's ability to support both novice and experienced users. Participants reported gains in efficiency, clarity, and comprehension, especially in understanding the structural logic of NCL. These findings support the broader hypothesis that AI-powered authoring environments can lower technical barriers and improve engagement in domain-specific authoring tasks.

Beyond its immediate application, this research contributes architectural and methodological principles that can be generalized to other DSLs. The layered authoring pipeline, modular agent architecture, and hybrid interaction model offer a transferable blueprint for developing intelligent authoring tools

in domains such as web development, game scripting, and educational content design.

A portion of this work has been published in the proceedings of WebMedia 2024, under the title *"Exploring Visual and Multimodal Interaction in NCL Authoring"*.

6.1 Limitations and Future Works

Despite its contributions, the proposed system faces notable limitations. Current large language models operate within fixed token windows, constraining the system's ability to handle large or complex NCL documents. Model hallucinations remain a concern, occasionally producing plausible but incorrect structures or fabricating non-existent elements. The multi-agent pipeline introduces latency and incurs financial cost due to multiple API calls, making it less suitable for real-time or resource-constrained environments.

On the visual side, the interface currently supports only spatial editing, omitting the temporal synchronization capabilities that are essential to NCL's full expressive potential. Voice and gesture recognition depend on environmental conditions and raise privacy concerns, particularly in professional settings. Furthermore, the tool's dependency on Visual Studio Code restricts its adoption to users within that ecosystem.

Methodologically, the evaluation was limited by sample size and controlled settings, which may not fully capture the diversity of real-world authoring contexts. Tasks were simplified to fit within practical study constraints, and longer-term adoption patterns remain unexplored. While user feedback was positive, it is unclear how much was influenced by the novelty of the system rather than sustained utility. Longitudinal studies will be essential to evaluate the impact of this tool on productivity, learning, and creative practices over time.

Several challenges are also inherent to the NCL language itself. Its temporal logic and context-sensitive constructs often require domain knowledge and intent clarification that are difficult to extract from natural language input alone. Simplifying the authoring process risks limiting users' ability to access the language's more advanced features. Additionally, reliance on external LLM services may pose privacy and security risks for organizations working with sensitive media content.

Future works should aim to expand the system's capabilities in several directions. Real-time visual previews and support for temporal editing would greatly enhance expressiveness and allow users to model more complex syn-

chronization patterns. Collaborative editing features could extend the tool's applicability to team-based environments and educational settings. Personalization mechanisms, such as adaptive agent behavior based on user profiles, would help tailor the authoring experience to individual preferences and levels of expertise. Moreover, enabling users to define persistent memory for agents, such as preferred code style, naming conventions, or structural patterns, could increase consistency across sessions and foster a sense of control over the generated output.

Furthermore, the use of agent orchestration frameworks and multi-agent management libraries may improve the accuracy of outputs. Extending support to other domain-specific languages would also demonstrate the generalizability of the proposed architecture. Finally, longitudinal studies are necessary to evaluate sustained adoption, evolving usability patterns, and the broader impact of such tools on authoring workflows over time. In parallel with these functional expansions, future research should also address the challenge of systematically evaluating the semantic quality of generated code. While syntactic validation ensures that outputs conform to NCL's grammar, it does not capture whether the generated structures faithfully represent the user's intent or adhere to domain-specific conventions. To address this, techniques such as reference-based semantic similarity metrics, human-in-the-loop expert annotation, and task-based evaluation protocols can be employed. Additionally, developing formal test suites or executable scenarios for generated NCL documents can provide a pragmatic measure of behavioral correctness.

At a broader level, this dissertation contributes to the discourse on AI-assisted creativity by demonstrating how large language models can augment, rather than supplant, human expertise. The proposed system reflects a design philosophy grounded in human agency, transparency, and collaboration. It underscores the potential of AI not merely to automate but to empower, transforming technical workflows into more inclusive, accessible, and creative processes.

Acknowledgments. This work was supported by RNP (Rede Nacional de Ensino e Pesquisa) and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior). The author extends deep gratitude to the volunteers who participated in the user study and to the advisors and collaborators whose insights and mentorship were essential to the successful development of this research.

6.2 Acknowledgments

This research was made possible through financial support from CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and RNP (Rede Nacional de Ensino e Pesquisa). The tool evaluation relied on the generous participation of volunteers who contributed their time and insights. The author gratefully acknowledges the guidance of their advisors and collaborators who made this work possible. Additionally, the author acknowledges the assistance of Large Language Models, which supported the translation of the text during the writing process.

7
Bibliography

ALLEN, J. F. Maintaining knowledge about temporal intervals. **Communications of the ACM**, ACM New York, NY, USA, v. 26, n. 11, p. 832–843, 1983.

AZEVEDO MARIO MEIRELES TEIXEIRA, C. d. S. S. N. Roberto Gerson de A. Ncl eclipse: Ambiente integrado para o desenvolvimento de aplicações para tv digital interativa em nested context language. In: **Salão de Ferramentas - SBRC 2009**. São Luís, MA, Brazil: [s.n.], 2009.

BANGOR, A.; KORTUM, P. T.; MILLER, J. T. An empirical evaluation of the system usability scale. **Intl. Journal of Human–Computer Interaction**, Taylor & Francis, v. 24, n. 6, p. 574–594, 2008.

BENDER, E. M. et al. On the dangers of stochastic parrots: Can language models be too big? In: **Proceedings of the 2021 ACM conference on fairness, accountability, and transparency**. [S.l.: s.n.], 2021. p. 610–623.

BHUIYAN, M.; PICKING, R. Gesture-controlled user interfaces, what have we done and what's next. In: CITESEER. **Proceedings of the fifth collaborative research symposium on security, E-Learning, Internet and Networking (SEIN 2009), Darmstadt, Germany**. [S.l.], 2009. p. 26–27.

BROWN, T. et al. Language models are few-shot learners. **Advances in Neural Information Processing Systems**, v. 33, p. 1877–1901, 2020.

BULTERMAN, D. C. Smil: Synchronized multimedia integration language. **MediaSync: Handbook on Multimedia Synchronization**, Springer, p. 359–385, 2018.

BULTERMAN, D. C.; HARDMAN, L. Multimedia authoring tools: State of the art and research challenges. **Computer science today: recent trends and developments**, Springer, p. 575–591, 2005.

CHEN, M. et al. Evaluating large language models trained on code. **arXiv preprint arXiv:2107.03374**, 2021.

CHEN, X. et al. Teaching large language models to self-debug. **arXiv preprint arXiv:2304.05128**, 2023.

COCKBURN, A. Hexagonal architecture. **Alistair Cockburn's blog**, 2005. Disponível em: https://alistair.cockburn.us/hexagonal-architecture/.

COSTA, I. V. et al. Ferramenta de validação ncl: Aprimorando o processo de desenvolvimento de aplicações de tv digital. In: **Anais Estendidos do XXX Simpósio Brasileiro de Sistemas Multimídia e Web**. Porto Alegre, RS, Brasil: SBC, 2024. p. 315–320. ISSN 2596-1683. Disponível em: https://sol.sbc.org.br/index.php/webmedia_estendido/article/view/30513.

EVANS, E. **Domain-driven design: tackling complexity in the heart of software**. [S.l.]: Addison-Wesley Professional, 2003.

GAMMA, E. et al. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Addison-Wesley Professional, 1995.

GAO, Y. et al. Retrieval-augmented generation for large language models: A survey. **arXiv preprint arXiv:2312.10997**, v. 2, 2023.

GUIMARÃES, R. L. Composer: um ambiente de autoria de documentos ncl para tv digital interativa. **Master's thesis, PUC-Rio**, 2007.

HALASZ, F.; SCHWARTZ, M. The dexter hypertext reference model. **Communications of the ACM**, ACM New York, NY, USA, v. 37, n. 2, p. 30–39, 1994.

HARDMAN, L.; BULTERMAN, D. C.; ROSSUM, G. V. The amsterdam hypermedia model: extending hypertext to support real multimedia. **Hypermedia**, Taylor & Francis, v. 5, n. 1, p. 47–69, 1993.

HUANG, L. et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. **ACM Transactions on Information Systems**, ACM New York, NY, v. 43, n. 2, p. 1–55, 2025.

ISHIBASHI, Y.; NISHIMURA, Y. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. **arXiv preprint arXiv:2404.02183**, 2024.

JAIN, J.; LUND, A.; WIXON, D. The future of natural user interfaces. In: **CHI'11 Extended Abstracts on Human Factors in Computing Systems**. [S.l.: s.n.], 2011. p. 211–214.

JOSHI, A. et al. Likert scale: Explored and explained. **British journal of applied science & technology**, v. 7, n. 4, p. 396–403, 2015.

KAUSHIK, D. M.; JAIN, R. Natural user interfaces: Trend in virtual interaction. **arXiv preprint arXiv:1405.0101**, 2014.

KHOT, T. et al. Decomposed prompting: A modular approach for solving complex tasks. **arXiv preprint arXiv:2210.02406**, 2022.

LEWIS, J. R. The system usability scale: past, present, and future. **International Journal of Human–Computer Interaction**, Taylor & Francis, v. 34, n. 7, p. 577–590, 2018.

LI, X. et al. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. **Vicinagearth**, Springer, v. 1, n. 1, p. 9, 2024.

LI, Y. et al. Competition-level code generation with alphacode. **Science**, American Association for the Advancement of Science, v. 378, n. 6624, p. 1092–1097, 2022.

LIU, X. et al. Dynamic multi-agent systems for task decomposition and iterative refinement. **arXiv preprint arXiv:2301.12345**, 2023.

LIU, Z. et al. A dynamic llm-powered agent network for task-oriented agent collaboration. In: **First Conference on Language Modeling**. [S.l.: s.n.], 2024.

MARTIN, R. C. **Clean architecture: a craftsman's guide to software structure and design**. [S.l.]: Prentice Hall Press, 2017.

MATTOS, D. et al. Assessing mulsemedia authoring application based on events with steve 2.0. **IEEE Access**, IEEE, 2025.

MATTOS, D. P. de; MUCHALUAT-SAADE, D. C. Steve: A hypermedia authoring tool based on the simple interactive multimedia model. In: **Proceedings of the ACM Symposium on Document Engineering 2018**. [S.l.: s.n.], 2018. p. 1–10.

MATTOS, D. Paulo de; SILVA, J. Varanda da; MUCHALUAT-SAADE, D. C. Next: graphical editor for authoring ncl documents supporting composite templates. In: **Proceedings of the 11th european conference on Interactive TV and video**. [S.l.: s.n.], 2013. p. 89–98.

MORAES, D. d. S. et al. On the challenges of using large language models for ncl code generation. In: SBC. **Anais Estendidos do XXIX Simpósio Brasileiro de Sistemas Multimídia e Web**. [S.l.], 2023. p. 151–156.

MORAES, D. d. S. et al. Lua2ncl: framework for textual authoring of ncl applications using lua. In: **Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web**. [S.l.: s.n.], 2016. p. 47–54.

MORENO, A. C. et al. Ncl 3.0: integrating new concepts to xml modular languages. **Multimedia Tools and Applications**, v. 50, n. 3, p. 345–364, 2010.

MORENO, M. F. et al. R&d progress on tv 3.0 application coding layer. **SET INTERNATIONAL JOURNAL OF BROADCAST ENGINEERING**, 2023. ISSN 2446-9432. Disponível em: https://dx.doi.org/10.18580/setijbe.2023.1.

MUTLU-BAYRAKTAR, D.; COSGUN, V.; ALTAN, T. Cognitive load in multimedia learning environments: A systematic review. **Computers & Education**, Elsevier, v. 141, p. 103618, 2019.

NIJKAMP, E. et al. Codegen: An open large language model for code with multi-turn program synthesis. **arXiv preprint arXiv:2203.13474**, 2022.

PAAS, F.; RENKL, A.; SWELLER, J. Cognitive load theory and instructional design: Recent developments. **Educational psychologist**, Taylor & Francis, v. 38, n. 1, p. 1–4, 2003.

RIEHLE, D. Composite design patterns. In: **Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**. [S.l.: s.n.], 1997. p. 218–228.

ROCHA, V. H. d. **DiTV–Arquitetura de desenvolvimento para aplicações interativas distribuídas para TV digital**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2013.

SANTOS, J. A. D. et al. A hybrid approach for spatio-temporal validation of declarative multimedia documents. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, ACM New York, NY, USA, v. 14, n. 4, p. 1–24, 2018.

SBTVD, F. **TV 3.0 project**. 2024. Disponível em: https://forumsbtvd.org.br/tv-3-0-project/.

SCHICK, T. et al. Toolformer: Language models can teach themselves to use tools. **Advances in Neural Information Processing Systems**, v. 36, p. 68539–68551, 2023.

SHEN, Y. et al. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. **Advances in Neural Information Processing Systems**, v. 36, p. 38154–38180, 2023.

SHINN, N. et al. Reflexion: Language agents with iterative self-improvement through reasoning traces. **arXiv preprint arXiv:2303.11366**, 2023.

SOARES, L. F.; MORENO, M. **Recommendation ITU-T H.761 (V3): Nested Context Language (NCL) and Ginga-NCL**. 2014.

SOARES, L. F. G. et al. Ginga: The brazilian digital tv middleware. **Journal of the Brazilian Computer Society**, v. 13, n. 3, p. 37–46, 2007.

SOARES, L. F. G. et al. Nested context language (ncl) and ginga: the brazilian digital tv middleware. **Proceedings of the IEEE**, v. 96, n. 1, p. 166–176, 2009.

SOARES, L. F. G.; RODRIGUES, R. F. Nested context language 3.0 part 8–ncl digital tv profiles. **Monografias em Ciência da Computação do Departamento de Informática da PUC-Rio**, v. 1200, n. 35, p. 06, 2006.

SOARES, L. F. G. S. **Programando em NCL 3.0: desenvolvimento de aplicaçoes para middleware Ginga: TV digital e Web**. [S.l.]: Elsevier, 2009.

TALEBIRAD, Y.; NADIRI, A. Multi-agent collaboration: Harnessing the power of intelligent llm agents. **arXiv preprint arXiv:2306.03314**, 2023.

VASWANI, A. et al. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

WANG, L. et al. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. **arXiv preprint arXiv:2305.04091**, 2023.

WANG, X. et al. Self-consistency improves chain of thought reasoning in language models. **arXiv preprint arXiv:2203.11171**, 2022.

WEI, J. et al. Emergent abilities of large language models. **arXiv preprint arXiv:2206.07682**, 2022.

WEI, J. et al. Chain-of-thought prompting elicits reasoning in large language models. **Advances in neural information processing systems**, v. 35, p. 24824–24837, 2022.

XI, Z. et al. The rise and potential of large language model based agents: A survey. **Science China Information Sciences**, Springer, v. 68, n. 2, p. 121101, 2025.

YAO, S. et al. Tree of thoughts: Deliberate problem solving with large language models. **Advances in neural information processing systems**, v. 36, p. 11809–11822, 2023.

YERRAMILLI, M.; VARMA, P.; DWARAKANATH, A. Multi-task pre-finetuning for zero-shot cross lingual transfer. In: **Proceedings of the 18th International Conference on Natural Language Processing (ICON)**. [S.l.: s.n.], 2021. p. 474–480.

ZHANG, A. **SpeechRecognition 2.1.3**. [S.l.]: PyPI - The Python Package Index, 2017. https://pypi.org/project/SpeechRecognition/2.1.3/. Accessed: 2024-08-20.

ZHANG, F. et al. Mediapipe hands: On-device real-time hand tracking. **arXiv preprint arXiv:2006.10214**, 2020.

ZHANG, Z. et al. Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. **Proceedings of the ACM on Software Engineering**, ACM New York, NY, USA, v. 2, n. ISSTA, p. 481–503, 2025.

ZHOU, J. et al. Large language models as agents. **arXiv preprint arXiv:2211.09110**, 2022.

A
Termo de Consentimento Livre e Esclarecido – Projeto "Ferramentas de Autoria de Aplicações NCL: Explorando Novos Métodos de Interação e o Uso de Inteligência Artificial"

A.1
Natureza da Pesquisa

Nós, pesquisadores responsáveis pelo projeto de pesquisa "Ferramentas de Autoria de Aplicações NCL: Explorando Novos Métodos de Interação e o Uso de Inteligência Artificial", sob coordenação do Professor Sérgio Colcher, do Departamento de Informática da PUC-Rio, lhe convidamos a participar como voluntário nesse estudo.

Nossa pesquisa visa a investigar diferentes formas de consumo e produção de visualização de dados. Envolve, entre outras coisas, entender como pessoas com diversos perfis e formações criam, buscam e utilizam diferentes visualizações para executar diferentes tarefas. O objetivo do estudo não é avaliar pessoas, mas sim o quão adequadas as visualizações e ferramentas são para a realização de certas tarefas. Através desta pesquisa espera-se identificar problemas e oportunidades de melhoria nos sistemas de apoio à visualização de informações.

Esta pesquisa envolverá as seguintes formas de captura de dados: entrevistas; sessões de observação de uso de software; questionários; e análise de logs de interação com software. Vale ressaltar que toda participação nesse estudo é inteiramente voluntária.

A.2
Benefícios

Os benefícios esperados envolvem um material didático aperfeiçoado sobre visualização de informação, artigos científicos sobre os diversos aspectos da pesquisa, e ferramentas de produção, busca e exploração de visualizações de informação. No entanto, não há benefícios a curto prazo esperados para os participantes do estudo.

A.3
Riscos e desconfortos

Identificamos alguns riscos mínimos associados à participação nesta pesquisa:

1. **Desconforto físico**: cansaço ou aborrecimento caso a sessão seja longa (acima de 2 horas). Vamos minimizar o tempo necessário à realização das

atividades, focando nas questões mais relevantes ao objetivo do estudo.

2. **Quebra da segurança digital** dos dados armazenados. Os dados coletados serão armazenados em ambiente seguro (mídia ou máquina sem acesso à internet ou em área protegida por senha). Além disto, o material coletado será desassociado da sua identidade, para garantir o seu anonimato e privacidade.
3. **Qualquer tipo de incômodo ou constrangimento**. Você pode interromper a pesquisa a qualquer momento e sem qualquer prejuízo, penalização ou constrangimento. Em nenhum lugar ficará registrado que você iniciou sua participação no estudo e optou por interrompê-la.

A.4 Garantia de anonimato, privacidade e sigilo dos dados

Esta pesquisa se pauta no respeito à privacidade, ao sigilo e ao anonimato dos participantes. Todos os dados brutos serão acessados somente pelos pesquisadores envolvidos nesta pesquisa e anonimizados para análise ou divulgação. O uso que faremos dos dados coletados durante o teste é estritamente limitado a atividades científicas, didáticas e de desenvolvimento de ferramentas que apoiem atividades de consumo e produção de visualizações de informação. Qualquer imagem, vídeo ou áudio divulgado será disfarçado para impedir a identificação dos participantes que nela aparecem.

A.5 Divulgação dos resultados

Os dados agregados e análises realizadas poderão ser publicados em publicações científicas e didáticas. Ao divulgarmos os resultados da pesquisa, nos comprometemos em preservar seu anonimato e privacidade, ocultando ou disfarçando toda informação (seja em texto, imagem, áudio ou vídeo) que possa revelar sua identidade. As informações brutas coletadas não serão divulgadas.

A.6 Acompanhamento, assistência e esclarecimentos

Todo material coletado será arquivado por no mínimo cinco anos. O Professor Sérgio Colcher, do Departamento de Informática da PUC-Rio, será a responsável por arquivar o material da pesquisa ao longo deste período e até o seu descarte. A qualquer momento, durante a pesquisa e até cinco anos após o seu término, você poderá solicitar mais informações sobre o estudo ou cópias dos materiais divulgados. Caso você observe algum comportamento que julgue antiético ou prejudicial a você, você pode entrar em contato para que sejam tomadas as medidas

necessárias. Ao final deste termo você encontra as formas de contato conosco ou com a Câmara de Ética em Pesquisa da PUC-Rio.

A.7
Ressarcimento de despesa eventual

Ao aceitar este termo, você não abre mão de nenhum direito legal. Se, por algum motivo, você tiver despesas decorrentes de sua participação nesse estudo, com transporte e/ou alimentação, você será reembolsado adequadamente pelos pesquisadores, conforme acordado no momento do recrutamento.

A.8
Liberdade de recusa, interrupção, desistência e retirada de consentimento

Sua participação nesta pesquisa é voluntária. Sua recusa não trará nenhum prejuízo a você, nem à sua relação com os pesquisadores ou com a universidade. A qualquer momento você pode interromper ou desistir da pesquisa, sem que incorra nenhuma penalização ou constrangimento. Você não precisará sequer justificar ou informar o motivo da interrupção ou desistência. Caso você mude de ideia sobre seu consentimento durante a sessão de estudo, basta comunicar sua decisão aos pesquisadores responsáveis, que então descartarão seus dados.

A.9
Consentimento

Eu, participante abaixo assinado(a), confirmo que:

1. Recebi informações detalhadas sobre a natureza e objetivos da pesquisa descrita neste documento e tive a oportunidade e esclarecer eventuais dúvidas;

2. Estou ciente de que minha participação é voluntária e posso abandonar o estudo a qualquer momento, sem fornecer uma razão e sem que haja quaisquer consequências negativas. Além disto, caso eu não queira responder a uma ou mais questões, tenho liberdade para isto;

3. Estou ciente de que minhas respostas serão mantidas confidenciais. Entendo que meu nome não será associado aos materiais de pesquisa e não será identificado nos materiais de divulgação que resultem da pesquisa;

4. Estou ciente de que a minha participação não acarretará qualquer ônus e que as atividades previstas na pesquisa não representam nenhum risco para mim ou para qualquer outro participante;

5. Estou ciente de que sou livre para consentir ou não com a pesquisa, conforme as opções que marco abaixo:

   Sobre a **coleta e uso de dados**:

☐ **Não autorizo** o uso das informações coletadas descritas neste documento.

☐ **Autorizo** o uso das informações coletadas conforme as condições descritas neste termo.

Rio de Janeiro, ___ de _____ de 20___

Pesquisador: _____ _____
(nome completo) (assinatura)

Participante: _____ _____
(nome completo) (assinatura)

Contatos: (1) Prof. Sérgio Colcher, Departamento de Informática, PUC-Rio - `colcher@inf.puc-rio.br` - Tel.: +55 21 3527-1500 ext. 4348; +55 21 98608-8974. (2) Câmara de Ética em Pesquisa da PUC-Rio: Rua Marquês de São Vicente, 225, Prédio Kennedy, 2º andar - Gávea - RJ. Tel.: +55 21 3527-1618