TU WIEN Informatics

Tracking User Interactions in Web-based Modeling Tools

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Dominik Granig, BSc
Matrikelnummer 11708471

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assoc. Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Wien, 11. Mai 2026

Dominik Granig	Dominik Bork

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Tracking User Interactions in Web-based Modeling Tools

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Dominik Granig, BSc
Registration Number 11708471

to the Faculty of Informatics

at the TU Wien

Advisor: Assoc. Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Vienna, May 11, 2026

Dominik Granig	Dominik Bork

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Dominik Granig, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 11. Mai 2026

Dominik Granig

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Dominik Bork, whose expertise, continuous support and patience have been invaluable throughout the completion of this thesis. The insightful feedback and encouragement helped me navigate the challenges of this work and grow as a researcher.

I would also like to thank my family, who supported me especially in the early stages of my studies and provided the foundation that made this journey possible. The support and encouragement I received during critical times gave me confidence to pursue my goals.

Finally, I want to thank my friends and fellow students Simon and David, who were among the very first to test this work. Their willingness to engage with the system and provide honest feedback was instrumental in shaping the final result.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Webbasierte Modellierungswerkzeuge haben in der Softwareentwicklung zunehmend an Bedeutung gewonnen, jedoch ist die Benutzerfreundlichkeit dieser Tools bislang wenig erforscht. Bestehende Analyse-Frameworks im Web sind nicht auf die semantischen Anforderungen von Modellierungsumgebungen ausgelegt, wodurch Entwicklerinnen und Entwicklern konkrete Daten zur Verbesserung der Benutzerfreundlichkeit und neuen Features fehlen.

Diese Arbeit beschreibt den Entwurf und die Implementierung eines offenen, modularen Frameworks zur Erfassung und Analyse von Benutzerinteraktionen in browserbasierten Modellierungswerkzeugen. Das Framework erfasst Interaktionen sowohl grundlegend als auch auf semantischer Ebene und integriert zudem eine eigenständige Blickerfassung (Eye-Tracking), um visuelle Aufmerksamkeit während des Modellierens aufzuzeichnen. Im Rahmen einer Design-Science-Research-Methodik werden drei Forschungsfragen behandelt: welche Interaktionen in grafischen Modellierungswerkzeugen relevant sind, wie erhobene Daten effizient strukturiert und analysiert werden können und wie Eye-Tracking in bestehende webbasierte Modellierungsumgebungen integriert werden kann.

Das entwickelte System besteht aus vier Komponenten: einer TypeScript-Frontend-Bibliothek zur Echtzeiterfassung von Interaktionsereignissen, einem eigenständigen Eye-Tracking-Modul auf Basis von WebGazer, einem Python/FastAPI-Backend mit REST-API zur Datenspeicherung und Analyse sowie einem PostgreSQL-Schema, das für zeitgestempelte Ereignisströme optimiert ist. Auf den erfassten Daten setzen drei Auswertungspipelines auf: N-Gramm-Clickstream-Analyse mit XES-Export, ein zusammengesetzter Friction-Score sowie Gauß-geglättete Blick-Heatmaps. Als Referenzimplementierung wurde das Framework in den BIGUML-Editor auf Basis der Graphical Language Server Platform (GLSP) integriert.

Eine explorative Fallstudie mit sieben Teilnehmenden zu einer UML-Klassendiagramm-Aufgabe brachte zwei qualitativ unterschiedliche Modellierungsstrategien zutage. Der vollständige Datensatz ist zur Reproduzierbarkeit auf Zenodo veröffentlicht. Das Framework zielt darauf ab, eine praktische Grundlage für empirische UX-Forschung und datengetriebene Verbesserungen von Modellierungseditoren zu bieten, unter Einhaltung der Prinzipien von Datenschutz, Datensicherheit und Datensparsamkeit.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Web-based modeling tools have become increasingly prominent in software engineering, yet the user experience (UX) of these tools remains largely unexplored. Existing behavioral analytics frameworks are not designed for the semantic nature of modeling environments, leaving developers without actionable data to guide usability improvements or develop new features.

This thesis captures the design and implementation of an open-source, modular interaction tracking framework for browser-based modeling tools. The framework captures user interactions at both the low-level input and high-level semantic action layers, and integrates a standalone eye-tracking component to record visual attention during modeling tasks. Following a Design Science Research methodology, the work addresses three research questions: which interactions are relevant in graphical modeling tools, how collected data can be efficiently structured and analyzed, and how eye-tracking can be embedded into existing web-based modeling environments.

The resulting system consists of four components: a TypeScript frontend library that captures real-time interaction events, a standalone eye-tracking module based on WebGazer, a Python/FastAPI backend exposing a REST API for data persistence and analysis, and a PostgreSQL schema optimized for timestamped event streams. On top of the captured data, three analytical pipelines are provided: n-gram clickstream analysis with XES export, a composite friction score, and Gaussian-smoothed gaze heatmaps. As a reference implementation, the framework was integrated into the BIGUML editor built on the Graphical Language Server Platform (GLSP).

An exploratory case study with seven participants on a UML class-diagram task surfaced two qualitatively different modeling strategies. The full dataset is published on Zenodo for reproducibility. The framework provides a practical foundation for empirical UX research and data-driven improvements to modeling editors, while addressing privacy, security, and data minimization principles.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Problem Statement and Motivation 1
1.2 Aim of the Thesis and Expected Results 2
1.3 Methodology 3

2 Related Work and Theoretical Background **7**
2.1 Foundations of User Interaction Tracking 7
2.2 Existing Monitoring Tools and Frameworks 8
2.3 Types of User Interactions 15
2.4 Data Analysis Techniques 15
2.5 Privacy and Ethical Principles 24
2.6 Synopsis 25

3 Requirement Analysis **27**
3.1 Functional Requirements 27
3.2 Non-Functional Requirements 29
3.3 Reference Integration Specifics 30

4 Framework Design and Implementation **31**
4.1 System Architecture Overview 31
4.2 Implemented Analytical Methods 36
4.3 Frontend Tracking Library 43
4.4 Eye-Tracking Integration 46
4.5 Interaction Replay 48

5 bigUML Integration **51**
5.1 bigUML Overview 51
5.2 Integration Approach 52

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.3 User Interface Integration . . . 52
5.4 Server Upload Integration . . . 53
5.5 Lessons Learned . . . 53
5.6 Challenges and Limitations . . . 54
5.7 Summary . . . 54

6 User Experiments and Data Collection 57
6.1 Experimental Setup . . . 57
6.2 Data Collection Process . . . 58
6.3 Analysis Approach . . . 59
6.4 Findings . . . 60
6.5 Discussion of Study Scope . . . 69
6.6 Evaluation Against Requirements . . . 70

7 Conclusion 73
7.1 Summary of Contributions . . . 73
7.2 Revisiting the Research Questions . . . 74
7.3 Limitations . . . 75
7.4 Future Work . . . 76
7.5 Closing Remarks . . . 76

Overview of Generative AI Tools Used 77

A Database Schema DDL 79
A.1 Recording Sessions Table . . . 79
A.2 Interaction Events Table . . . 80
A.3 Gaze Points Table . . . 80

B Frontend Tracking Library Code 83
B.1 InteractionTracker Service Initialization . . . 83
B.2 Interaction Event Type Enumeration . . . 84
B.3 GLSP Action Listener Registration . . . 85
B.4 GLSP Action Processing . . . 85
B.5 Debounced Viewport Tracking . . . 86
B.6 ViewportTrackingHandler in GLSP Client . . . 87
B.7 ElementBoundsTrackingHandler . . . 87
B.8 Session Export Implementation . . . 88
B.9 Diagram Type Detection . . . 89

C Eye-Tracking Integration Code 91
C.1 Standalone Eye-Tracking Application Structure . . . 91
C.2 WebGazer Initialization . . . 92
C.3 Calibration Point Handling . . . 92
C.4 Gaze Data Export . . . 93

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

C.5 Eye-Tracking Launch Command 94

D Interaction Replay Code **95**
D.1 ID Pre-Mapping During Event Analysis 95
D.2 Runtime ID Capture from Server Responses 96
D.3 ID Translation for Edge Creation 97
D.4 Container Tracking for Element Creation 97
D.5 Speed-Controlled Replay 98

E bigUML Integration Code **101**
E.1 Package Dependency Configuration 101
E.2 Module Registration in VSCode Extension 101
E.3 GLSP Client Module for Tracking 102
E.4 Eye-Tracking Provider Setup 102
E.5 Session Upload to Server 102

F Integration Guide for Third-Party Modeling Frontends **105**
F.1 The Backend Contract 105
F.2 Fixed Fields vs Flexible JSONB 106
F.3 Mapping Tool-Native Operations to Canonical Event Types 110
F.4 Implementation Recipe 111
F.5 Optional: Eye-Tracking Integration 114
F.6 Validation Checklist 114

List of Figures **117**

List of Tables **119**

Bibliography **121**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

This chapter outlines the motivation behind this thesis and introduces the central research questions it addresses. It also provides an overview of the methodological approach taken and briefly summarizes the structure of the remaining chapters.

1.1 Problem Statement and Motivation

Web-based modeling tools have made software modeling more accessible by lowering technical barriers and enabling modeling directly in the browser. At the same time, new innovative frameworks such as GLSP - while making use of standard web technologies - have lowered the barrier for developers, allowing them to easily build and expand ever-growing ecosystems of modeling tools packed with innovative features [BLO23]. However, user experience (UX) of modeling tools still remains to be explored in detail [KL22].

Web-based modeling frameworks such as GLSP primarily focus on providing infrastructure for implementing graphical modeling editors, while dedicated facilities for systematically recording and analyzing modeler interactions are typically provided by specialized research tools such as TOOL or MER [MB25, TRSB20, DBW23].

Newly introduced software features are not necessarily adopted or used as intended. Prior research has shown that discoverability and perceived value can significantly influence users' adoption decisions [BA22]. However, without concrete usage data it can be difficult for developers to determine whether limited adoption results from usability issues, insufficient discoverability, or a mismatch between user expectations and the provided functionality.

Having an option to capture and analyze the interaction patterns of users during the modeling process, including creating, deleting, dragging and resizing elements, and especially visual attention (achieved via eye tracking), could help tool-providers to

improve usability, optimize their interfaces, and contribute towards more user-centric and model-based software engineering [BS14].

While session recording and behavioral analytics frameworks such as PostHog [Pos26], FullStory [Ful26], or Hotjar [Hot26] offer general-purpose tracking solutions for traditional web applications, they are either proprietary, limited in how to configure them or simply not designed with the particular requirements of modeling environments and their semantic user actions in mind.

A notable exception in the research landscape is *TOOL*—a web-based modeling observatory designed for studying individual modeling processes in a controlled, research-driven context [TRSB20]. TOOL provides a "multi-modal" data collection setup, including UI event tracking, screen recordings, verbal protocol capture during modeling, and surveys. However, its current implementation is tied to specific use cases (e.g., ER and BPMN modeling) and focuses on controlled lab studies rather than offering a lightweight integration into real-world frameworks or modeling platforms.

1.2 Aim of the Thesis and Expected Results

This thesis focuses on the design and implementation of an open-source framework for capturing and analyzing user interactions in browser-based modeling tools. The framework aims to be **modular**, **extensible**, and **independent** of a specific modeling tool, allowing it to be reused across different modeling environments. This would enable a low-effort integration into a wide range of modeling environments while still supporting the capture and evaluation of multiple forms of user interactions. Ultimately, this framework aims to be used to support both empirical UX research and practical modeling-editor improvements.

To demonstrate the applicability of such a framework in a real-world context, the Graphical Language Server Platform (GLSP) is used as a reference integration. GLSP serves as a reasonably representative and modern web-based modeling platform, and its open architecture makes it well-suited for showcasing how interaction tracking can be embedded into domain-specific graphical editors.

In particular, the expected outcomes of this thesis include:

- **A modular JavaScript/TypeScript-based tracking library (frontend)**, developed as part of the framework and compatible with web-based modeling platforms. The library will handle the communication with the Graphical Language Server Platform (GLSP) and provide built-in support for eye-tracking technologies. It will communicate its findings to the backend for analysis.
- **A general-purpose interaction tracking framework (backend)**, designed to be lightweight, extensible, and tool-agnostic. It will support capturing various user actions—such as element selections, gaze-points (from eye-tracking), and keyboard inputs—along with the high-level semantic events relevant to modeling tasks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- **An instrumented GLSP-based demonstrator tool**, realized by integrating the tracking framework into BIGUML—a web-based UML editor developed according to the GLSP reference architecture proposed by Metin and Bork [MB25]. This demonstrator will serve as a realistic and modular environment for evaluating the capabilities of the tracking framework.
- **Small-scale evaluation:** A small-scale user study using the demonstrator tool to collect and analyze recorded user interactions in a controlled setting. Participants perform representative modeling tasks while their actions and eye movements are logged in detail. The resulting dataset is evaluated to capture all relevant interaction events as well as the resulting higher-level semantic actions (e.g., sequence of creating or modifying model elements). The data from this case study aims to illustrate the framework's core ability and enable empirical analysis of user behavior in future works.

According to the objectives and scope of this work, the following research questions are being addressed over the course of this thesis. These questions are intended to guide the design, implementation, and evaluation of the proposed interaction tracking framework and its integration into web-based modeling tools:

(**RQ1**) What user-interactions are of importance in a graphical modeling tool and ought to be collected?

(**RQ2**) To what extent can collected interaction data be structured, stored, and analyzed to provide actionable insights for improving user experience?

(**RQ3**) To what extent can eye-tracking be effectively integrated into existing web-based modeling tools using different tracking technologies?

While not a main research question, the framework design will incorporate essential security and privacy principles to ensure responsible data collection and user consent handling.

1.3 Methodology

The methodology of this thesis follows a structured process inspired by Design Science Research (DSR), consisting of four phases that directly address the defined research questions [HMJR04]. The process encompasses the analysis of earlier related work, the design and implementation of an artifact (i.e., the tracking framework), and its empirical evaluation in a realistic use case setting.

This mixed-method approach combines qualitative and quantitative methods to both guide the development of the tracking system and evaluate its usability and technical impact.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

1. **Literature Review and Requirements Analysis (RQ1)** This initial phase focuses on establishing the theoretical foundations needed for designing the tracking framework by reviewing and synthesizing relevant literature:
   - Conducting a literature review on user interaction tracking in web-based tools, with a focus on modeling environments and diagram-based editors.
   - Extracting key types of interactions that are commonly monitored (e.g., mouse events, drag-and-drop operations, command executions, gaze fixations).
   - Identifying domain-specific requirements for modeling tools, including the importance of tracking semantic actions (e.g., “create element x” vs. just a simple “click”).
   - Reviewing literature on privacy, security, and ethical handling of user interaction data, especially in the context of eye tracking and behavioral analytics.
2. **Framework Design and Implementation (RQ1, RQ3)** Following the DSR methodology, the tracking framework is designed and implemented iteratively:
   - Architectural design for a modular, extensible tracking framework, using established software architecture patterns such as separation of concerns and plugin-based extensibility.
   - Compatibility with the Graphical Language Server Platform (GLSP) by building the core tracking functionality in JavaScript/TypeScript.
   - Develop connector for different eye-tracking technologies and define a unified data model for gaze event handling.
   - Annotate user actions with semantic meaning where possible and define a schema for structured logging of interaction data.
3. **Evaluation Setup and Data Collection (RQ2, RQ3)** To validate the utility and performance of the framework, an empirical evaluation will be conducted based on principles of user-centered design and exploratory usability testing:
   - Integrate the framework into the BIGUML tool to serve as a realistic testbed for the whole system.
   - Design and conduct controlled user sessions with voluntary participants, capturing both interaction and gaze data.
   - Develop a "replay" feature that re-creates a model step by step solely on the collected data to easily confirm that the system can assign all tracked actions the right semantics.
   - Apply established techniques for visualizing and analyzing the collected data, including heatmaps, and clickstream mapping.
4. **Security and Privacy Review (supporting all RQs)** Ethical and responsible data handling is a cross-cutting concern and will be addressed in a dedicated review phase:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Evaluate user consent mechanisms, in compliance with common privacy frameworks (e.g., GDPR-style opt-in behavior).
- Analyze local vs. remote data storage strategies and their implications for data ownership and control.
- Ensure the tracking system allows for user transparency (e.g., view logged data) and revocability (opt-out or delete sessions).

This methodology ensures that both theoretical and practical aspects of interaction tracking in web-based modeling tools are addressed, leading to a validated and reusable framework that supports future UX research and tool development.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Related Work and Theoretical Background

This chapter explores existing literature from human-computer interaction (HCI), visual analytics, and modeling tool design to establish the theoretical background for the proposed interaction tracking framework. The main goal is to identify existing tools and frameworks, determine which types of user interactions might be meaningful in these modeling environments, what analysis techniques can be applied to the captured data and what privacy and ethical principles must be considered.

2.1 Foundations of User Interaction Tracking

This section will give a general overview of user interaction tracking, including common methodologies and challenges in order to give some context before focusing on the specifics of modeling environments.

User Behavior Analytics (UBA), commonly found in cybersecurity, focuses on analyzing user activity patterns and tends to overlap conceptually with interaction tracking approaches in modeling environments. UBA essentially tries to analyze user activity to find "unusual patterns" that could indicate security threats. It does this by collecting and analyzing data from various sources, such as login times, file access, and application usage to establish a baseline. Certain frameworks then continuously monitor user activity and flag deviations. Recent evolutions in this field talk about UEBA (User and Entity Behavior Analytics) which essentially also includes other entities like Programs and Servers for monitoring.

User behavior tracking and Recommendation Engines are terms used in marketing. User behavior tracking fundamentally involves the systematic recording of user actions,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

such as clicks, scrolls, and form submissions, to identify patterns and pinpoint areas of interest or difficulty. This process reveals the user's journey through a website or application and can support the recommendation of personalized and engaging "content".

Widely adopted methods in these industries include:

- **Session Replays:** These are video-like playbacks that provide a precise visual record of how users navigate and engage with a site. They can offer contextual information often missing from numerical data alone, hinting at the underlying reasons for user actions and assisting in identifying usability issues and user struggles [Kha25].
- **Heatmaps:** These tools generate aggregated, color-coded visualizations that illustrate where users click, tap, scroll, and move on a webpage. They are highly effective in highlighting popular features, identifying neglected areas, and pinpointing potential points of user frustration, such as "rage clicks" (repeated clicks indicating frustration) [Kha25].
- **Funnel and Path Reports:** These analytical tools visualize the customer journey through predefined stages, enabling the identification of drop-off points, common user flows, and the "sunny day path" that leads to successful outcomes. They help in understanding the sequence of events involved in achieving the overall goals of applications [Kha25].
- **Surveys and Feedback Mechanisms:** While not direct tracking of actions, on-site surveys and feedback tools are widely used for collecting qualitative metrics, such as user loyalty, usability perceptions, and overall satisfaction. They provide direct user sentiment and suggestions for product improvement [Kha25].
- **Event Segmentation:** This involves the tracking and visualization of user interactions as discrete events, facilitating granular analysis of specific actions performed within the product [Kha25].
- **Eye Tracking:** Though commonly used in lab environments, specialized eye-tracking software, utilizing standard webcams or specialized equipment to monitor eye movements and gaze fixations. This technology reveals user attention, engagement with visual elements, and scanning patterns on a screen [PHCJ25].

2.2 Existing Monitoring Tools and Frameworks

A number of tools and studies have explored interaction logging beyond traditional web analytics, each targeting a different point in the design space between raw event capture and actionable behavioral insight. On one end sit commercial platforms like Hotjar [Hot26] and FullStory [Ful26], which prioritize ease of deployment and breadth of capture over configurability or data ownership. On the other end sit purpose-built

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

research instruments like Interaction Lab [FLDAGRT22] and WebTrace [PHCJ25], which trade general-purpose usability for precision in specific study contexts. Somewhere in between, open-source platforms like PostHog [Pos26] attempt to give development teams the depth of a commercial tool without the dependency on third-party infrastructure. The following overview examines each of these tools in turn, with a particular focus on what they capture, how they have been used or evaluated in the literature, and where they fall short for the specific demands of interaction tracking in diagram-based modeling environments.

PostHog. PostHog [Pos26] is an open-source, self-hostable product analytics platform that targets developer teams rather than marketing departments. Its autocapture mechanism instruments frontend events – clicks, pageviews, form submissions, rage clicks – without requiring manual event tagging for each interaction, which significantly lowers the barrier to getting something running. Beyond raw event capture, PostHog ships session replay, user path analysis, conversion funnels, feature flags, and A/B testing as a single integrated stack built on top of ClickHouse [Cli26]. The self-hosting option is the main reason it tends to come up in research and privacy-sensitive contexts: data never leaves the operator's infrastructure, which matters both for GDPR compliance and for studies where exporting interaction logs to a third-party cloud is not acceptable. The tradeoff is operational overhead – running and maintaining a ClickHouse-backed deployment is non-trivial, and the platform is explicitly aimed at engineers rather than general users. For the purposes of this thesis, PostHog is interesting as a reference point for what a full-featured, self-hosted event tracking pipeline looks like, but it is not designed for the semantic granularity that modeling environments require: it captures DOM events, not modeling operations.

WebTrace. WebTrace [PHCJ25] is a more recent and narrower tool, introduced in 2025 specifically to support remote usability testing. It captures click events, navigation transitions, and text input during web-based task sessions and focuses on low-overhead deployment for study scenarios where participants work unsupervised in their own browser. The key design goal is reproducibility of interaction sequences rather than real-time dashboarding, which sets it apart from platforms like PostHog that prioritize aggregate behavioral analytics. WebTrace is most relevant here as an example of the lightweight, study-oriented end of the spectrum – purpose-built for a narrow capture task with minimal infrastructure requirements. Like the other generic tools, it operates on raw DOM events and has no notion of higher-level modeling operations such as creating a node or establishing a typed connection.

Hotjar. Hotjar [Hot26] is a commercial behavior analytics platform that has been widely adopted for website UX work, offering heatmaps, session recordings, feedback polls, and form analytics behind a JavaScript snippet that requires no backend changes from the developer. In practice it is used to complement quantitative analytics (e.g., from Google Analytics) with qualitative session data: watching recordings of confused users

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

or inspecting click heatmaps is often faster than hypothesizing from aggregate numbers alone. That said, Hotjar has attracted persistent criticism in the privacy literature. Acar et al. [AEN20] identified Hotjar among seven session replay services found on hundreds of high-traffic sites and showed that automated redaction of sensitive form fields was incomplete across all providers studied – typed content from non-standard input elements leaked to third-party servers despite nominal opt-out signals. Polčák and Slezáková [PS23] followed up specifically on Hotjar and demonstrated that even after the service's stated fixes, exfiltration of page content to Hotjar's servers remained possible under certain conditions, and that the Do Not Track HTTP header was not reliably respected. These issues are directly relevant for any deployment in contexts where participants enter personal or sensitive data, which is a realistic concern in study settings.

FullStory. FullStory [Ful26] occupies the enterprise end of the session replay market, positioning itself on the depth of its indexing: every user interaction is automatically captured and stored in a format that can be replayed or queried later without requiring pre-defined event schemas. This "capture everything" approach is genuinely useful for debugging unexpected user behavior after the fact, since the session log exists even for interaction sequences nobody thought to instrument in advance. Grodzinsky et al. [GMW22] analyzed FullStory directly as a case study in session replay privacy, applying Nissenbaum's contextual integrity framework to argue that the scope of data collected routinely exceeds the stated purpose of usability improvement, and that users have no meaningful way of knowing when and how their sessions are being forwarded to third-party infrastructure. The same Princeton study by Acar et al. [AEN20] found FullStory scripts deployed on major commercial sites and confirmed that medical and personal information leaked through imperfect redaction even on sites that had configured the service's privacy settings. For research use, FullStory's closed, cloud-only architecture means there is no path to self-hosting and no programmatic access to raw event logs in a format suitable for offline analysis – which makes it unsuitable as an interaction logging backend for a scientific study requiring full data ownership and reproducibility.

Interaction Lab. Interaction Lab [FLDAGRT22] takes a different angle from the tools above. Rather than capturing page-level navigation and click paths, it focuses on low-level pointer dynamics: mouse speed, acceleration, trajectory curvature, correction counts, and similar kinematic features that are routinely discarded by mainstream analytics tools. The rationale is that these signals carry information about cognitive effort and hesitation that coarser event logs miss entirely – a slow, curved path to a button tells a different story than a direct click even if the logged event is identical. This granularity is particularly relevant for modeling environments, where hesitation before placing a notation element or repeated backtracking on the canvas may indicate genuine difficulty with the modeling task rather than general UI friction. Interaction Lab was evaluated in the context of web application usability and demonstrated that its kinematic features could distinguish between different levels of user difficulty in ways that click-level event logs could not. However, the tool was designed as a standalone research instrument

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

rather than a library, and like the other generic tools it has no built-in notion of the structured, language-specific operations that characterize diagram-based modeling.

2.2.1 TOOL: A Research Observatory for Conceptual Modeling

While the tools surveyed in the previous section cover general-purpose interaction tracking well, none of them were built with the semantics of diagram-based modeling in mind. TOOL, developed at the Enterprise Modeling Research Group of FernUniversität in Hagen since 2013 [TS18], is the most closely related prior work to this thesis. It is a web-based modeling tool and research observatory whose explicit goal is to study how individuals perform conceptual modeling, where they struggle, and how targeted tool support could help [Ter17, TRBS20].

Architecture and Modeling Editors. Technically, TOOL is a standard web application: a JavaScript-driven browser frontend communicates with a Java EE application server backend over HTTPS [RTS20]. The frontend currently ships with two graphical modeling editors. The first targets data modeling using an adapted variant of the Entity-Relationship Model (ERM), and the second covers business process modeling with a subset of BPMN 2.0. Both editors are backed by stencil sets that encode the abstract and concrete syntax of the respective modeling language, including explicit typing, connection rules, and visual appearance. This separation between language definition and editor runtime is conceptually similar to the grammar-based approach used in modern language workbenches, though TOOL's stencil sets are JSON-based and evaluated at runtime rather than compiled.

On top of the editors, TOOL provides two forms of in-editor support for modelers. First, ad-hoc syntax validation checks models against the connection rules and cardinality constraints of the active stencil set and highlights violations directly on the canvas. Second, a Natural Language Processing (NLP) component, built on the Stanford CoreNLP toolkit, analyzes the natural-language task description provided to a modeler and suggests candidate labels for entity types, relationship types, and attributes as the model is being constructed. This feedback component has been reported as useful in informal evaluations, though a controlled assessment of its effect on modeling outcomes could not be found at the time of writing.

Multi-Modal Observation Infrastructure. What sets TOOL apart from a plain modeling editor is its integrated research observatory. Rather than relying on a single data source, TOOL combines four complementary observation modes that can be switched on or off per study and presented to participants in a configurable sequence [RTS20].

The first and most fundamental mode is *interaction event tracking*. Every user action on the canvas — adding a node, drawing an edge, relabeling an element, deleting, undoing — is recorded as a timestamped discrete event and stored in a relational database. This produces a complete, ordered log of the modeling session that can later be analyzed

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

statistically. Tracked data is visualized in three complementary ways: step-by-step and automatic replays that reconstruct the construction sequence on the canvas, dot diagrams that plot event types against normalized time (inspired by the PPMCharts of Claes et al. [RTS20]), and heatmaps generated from mouse-movement and click coordinates that reveal which parts of the canvas and palette received the most attention.

The second mode is *think-aloud verbal protocol recording*. Participants are asked to verbalize their reasoning while working, and a browser-based audio recorder captures the audio stream as a WAV file. A visual reminder prompts the participant to keep talking if they go silent for too long. Verbal protocols are subsequently coded against the audio-visual recording in qualitative data analysis software (MAXQDA was used in the published studies) using cognitive breakdowns [RTS20] — moments where a modeler's line of reasoning visibly fails — as the primary analytic category.

Third, TOOL supports *screen capture* via WebRTC, producing a video recording of the screen that can be time-synchronized with the verbal protocol. This is mainly used in lab settings where participants model on-site, and it allows analysts to catch relevant interactions outside the editor (e.g., consulting a printed task description or reaching for paper and pen).

Finally, the observatory includes a *survey component* that can present closed- and open-ended questionnaires to participants before and after a modeling session. Survey responses are linked to the corresponding interaction log and conceptual model in the database, making it straightforward to correlate self-reported difficulties with observable behavior patterns.

A configurable observation workflow ties these four modes together. Researchers can select which modes to activate, define the sequence of steps participants walk through (consent form, video introduction to the tool, pre-modeling survey, modeling task, post-modeling survey), and hide or expose specific UI elements of the editor itself — for example, disabling the syntax checker to study how modelers behave without real-time feedback [TRBS20].

Evaluation and Deployment. TOOL has been evaluated in two exploratory lab studies. The first, conducted in January 2019 with eight learners of conceptual modeling, applied the full multi-modal observation setup to a data modeling task drawn from a library domain [RTS20]. Analysis of the combined verbal protocols, interaction logs, and post-modeling surveys led to the identification of five recurring types of modeling difficulty, with the majority clustering around relationship types: specifically, deciding whether a concept warrants an entity type or a relationship type, constructing meaningful identifiers for relationship types, and determining cardinalities. A second study in May–June 2019 observed eight experienced modelers on the same task to compare how difficulties manifested across expertise levels; analysis was ongoing at the time of the last published update [TRBS20].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Beyond the lab, TOOL has been running in production as part of introductory university courses at FernUniversität, serving over 200 students per semester. This larger deployment has been used to stress-test runtime stability under concurrent load and to validate the tracking infrastructure at scale. The tool has also been used at the Universitat Politècnica de València and KU Leuven.

Relationship to This Thesis. TOOL demonstrates convincingly that combining a graphical modeling editor with a structured interaction tracking backend is both technically feasible and scientifically productive. However, it was built for a specific research agenda and correspondingly makes several assumptions that limit its applicability as a general-purpose tracking layer. The graphical editors support exactly two modeling languages, and adding a third requires changes inside TOOL's core codebase. The observation infrastructure is tightly integrated with those editors rather than being exposed as a reusable library that an external tool could connect to. The tracking event model reflects the specific operations of TOOL's own canvas and is not aligned with any standardized interface for modeling tools. Finally, the primary target use case is controlled lab studies, and the deployment model — a monolithic Java EE application that is not accessible by the public — is not well-suited for lightweight integration into open-source modeling frameworks developed by third parties.

These constraints directly motivate the design goals of the framework presented in this thesis: a modular, tool-agnostic tracking library that exposes a standardized event interface, can be dropped into existing web-based modeling environments with minimal effort, and is independent of any particular modeling language or editor implementation.

2.2.2 MER: Modeling Event Recorder

Dehghani et al. [DBW23] present MER (Modeling Event Recorder), a tool for capturing and analyzing user interactions in graphical modeling editors. The work is motivated by the same observation that drives this thesis: understanding how modelers actually use graphical editors – what they create, delete, and modify, and in what order – is valuable both for improving the editors themselves and for studying the quality of the resulting models. MER targets EMF-based tooling and supports two families of editors: Sirius-based desktop editors running inside the Eclipse IDE, and web-based editors built on the Graphical Language Server Platform (GLSP), making it directly relevant to the integration scenario explored in this thesis.

Architecture and Event Capture. The core of MER is a listener that hooks into the EMF Notification API [DBW23]. Whenever a modeler modifies a model through the editor GUI – creating an element, setting an attribute, drawing a connection, or deleting something – EMF fires a notification object describing the change. MER intercepts these notifications, attaches timestamps and session metadata, and accumulates them into a structured event log. For Sirius-based desktop editors, MER is packaged as an Eclipse plugin and is available on the Eclipse Marketplace. For GLSP-compliant web

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

editors, the equivalent component is deployed server-side as a Java application, since in the GLSP architecture all modeling logic is handled by the server and the browser client is intentionally kept thin.

At the end of a session, MER serializes the collected events into three complementary artifacts: an IEEE XES log file, an EMF-based XES model conforming to a custom XES metamodel, and an extended EMF Change Model. The XES file is the primary output for analysis, as it is accepted by a number of process mining tools, including the open-source ProM framework [DBW23]. By piping MER's output into ProM, analysts can apply standard process discovery algorithms – inductive miner, ILP miner, heuristic miner – to reconstruct a process model that describes how modelers navigated the editor, and from that identify common patterns or bottlenecks.

Evaluation. MER has been applied in a range of contexts. Initial experiments covered Ecore modeling with Eclipse Ecore Tools and produced process models that visually showed the typical create-set-name lifecycle of metamodel elements. Beyond these controlled examples, MER was deployed as part of an industrial case study with Volvo Construction Equipment within the AIDOaRt European project [DBW23], where interaction logs from AutomationML and SysML editors were collected and analyzed to inform tool improvements. This combination of academic prototype and industrial deployment gives MER stronger external validity than most research tools in this space.

Relationship to This Thesis. MER and the framework presented in this thesis share the same high-level motivation – recording what happens inside a graphical modeling editor – but operate at different layers and with different goals. MER works at the *model layer*: it intercepts EMF notifications that describe changes to the underlying model object graph, which means it captures exactly the modeling operations the editor considers semantically meaningful (create node, set attribute, delete edge), but nothing else. UI-level interactions that do not immediately result in a model change – mouse movements, hover dwell, zoom and pan gestures, keyboard shortcuts, failed drag attempts, or gaze fixations – are invisible to MER by design. This makes MER well-suited for process mining and quality analysis of modeling outcomes, but it cannot support the kind of UX friction analysis or attention heatmaps that require capturing the full stream of low-level input events. The framework in this thesis is explicitly designed to capture that lower layer, complementing MER's model-centric approach with a UI-centric one. In addition, MER is tied to the EMF ecosystem: extending it to a non-EMF modeling platform would require a different notification mechanism and a different deployment strategy, whereas the framework proposed here aims to be designed tool-agnostic.

2.2.3 Comparative Overview

To synthesize the discussion above, Table 2.1 contrasts the surveyed tools along five dimensions that matter for the framework proposed in this thesis: licensing, deployment model, the granularity of captured interactions, the data analysis techniques supported

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

out of the box, and whether the tool exposes any awareness of the semantic operations of a graphical modeling editor rather than only generic web events. The two extremes of the design space stand out clearly. Commercial session-replay platforms such as Hotjar and FullStory offer the most polished analysis features but are cloud-only, proprietary, and treat every application as an unstructured DOM. Research-oriented and open-source tools such as Interaction Lab, WebTrace, and PostHog return control over deployment and data ownership at the cost of feature breadth or maturity. Only TOOL and MER explicitly capture the semantic layer of a graphical modeling editor, and both do so under assumptions – fixed languages in TOOL, EMF-only model events in MER – that are too restrictive to serve as a general-purpose tracking layer for arbitrary GLSP-based modeling tools.

2.3 Types of User Interactions

Several works identify sets of user interactions that are particularly relevant for understanding behavior in diagram-based modeling tools. These interactions include mouse-based actions such as clicking, dragging, hovering, and selection paths [ZZVW25, SAAKW25], as well as the use of keyboard shortcuts, which are commonly employed to accelerate modeling tasks. Beyond these low-level inputs, researchers emphasize the importance of capturing and mapping semantic actions, i.e., domain-specific operations such as creating nodes, deleting connections, or renaming elements, as they more directly reflect user intent during the modeling process [CS08]. In addition, interactions related to view manipulation—such as zooming, panning, and collapsing or expanding parts of a model—provide insight into navigation and spatial orientation strategies [MDCSW25]. Finally, eye-gaze behavior has been shown to be a valuable source of information for inferring user focus, attention shifts, and aspects of cognitive load during modeling tasks [SAAKW25, AATWL25].

Collectively, these interaction types go beyond raw UI events and by capturing them enable a more meaningful analysis of modeling behavior and underlying cognitive strategies. The table below presents an aggregated overview of the identified interaction types, corresponding capture mechanisms, and their potential interpretations.

2.4 Data Analysis Techniques

This section focuses on a selected set of analysis techniques that can directly be applied within the scope of the envisioned framework. Rather than aiming for a comprehensive overview of all possible behavioral analysis methods, the emphasis is placed on techniques that are practical, reproducible, and well-suited to browser-based modeling environments. The explored approaches primarily build on sequential interaction analysis, spatial attention visualization, and interaction-based heuristics for identifying potential usability issues. In particular, the analysis covers clickstream-based methods, including n-gram analysis, fixation heatmap visualization derived from gaze data, and pattern-based

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 2.1: Comparative overview of existing monitoring tools and frameworks.

Tool	**License**	**Deployment**	**Interaction Capture**	**Data Analysis Techniques**	**Modeling-aware?**
PostHog [Pos26]	Open source (MIT; `ee/` proprietary)	Self-host or SaaS cloud	Autocapture of generic DOM events: clicks, pageviews, form submissions, rage clicks; manual instrumentation possible	Session replay, funnels, user paths, retention, heatmaps, A/B tests, feature flags, SQL/HogQL on ClickHouse	No
WebTrace [PHCJ25]	Source available (GitHub, no explicit licence)	Chrome extension + Firebase-backed web dashboard	Mouse clicks, keyboard inputs, page navigation, scrolls; client-side via extension, no host application changes	Behaviour abstraction (collapses repetitive sequences), per-session inspection in dashboard, CSV-style export	No
Hotjar [Hot26]	Proprietary (commercial SaaS)	Cloud-only	Clicks, mouse movement, scroll, keystrokes (with masking), form interactions; via JS snippet	Click / move / scroll / engagement-zone heatmaps, session recordings, funnels, surveys, frustration signals (rage / dead / error clicks)	No
FullStory [Ful26]	Proprietary (commercial SaaS)	Cloud-only	Tagless autocapture of all DOM interactions: clicks, scrolls, form inputs, navigation, tab switches; web and mobile SDKs	High-fidelity session replay with retroactive search, funnels, user journeys, frustration signals, AI session summaries	No
Interaction Lab [FLDAGRT22]	Research prototype	Self-hosted (web client + collection agent + analysis dashboard)	Low-level pointer kinematics: mouse speed, acceleration, trajectory curvature, correction counts; complements raw click events	Plugin-based feature extractors (Strategy pattern), CSV export of raw and computed features for offline statistical analysis	No
TOOL [RTS20]	Closed (research prototype)	Self-hosted Java EE application (monolithic)	Multi-modal: semantic canvas events (add / relabel / delete node, draw edge, undo), think-aloud audio (WAV), screen capture (WebRTC), pre/post surveys, mouse coordinates	Step-by-step and automatic canvas replay, PPM-style dot diagrams, mouse/click heatmaps, qualitative coding (MAXQDA), survey integration	Yes (ER, BPMN only)
MER [DBW23]	Open source (EPL 2.0)	Eclipse plugin (Sirius) or Java server component (GLSP)	Model-level events via the EMF Notification API: create, set, delete operations on the underlying model graph	XES log export (IEEE 1849-2016), EMF-based XES models, Change Models; process mining via ProM (inductive, ILP, heuristic miners)	Yes (EMF/Sirius, GLSP)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Table 2.2: Interaction Types, Events, Tracking Methodologies, and Interpretation Potential

Interaction Type	**Event Description**	**Tracking Methodologies**	**Interpretation**
Mouse Click	mousedown, mouseup, click events, coordinates, DOM element ID	Event Listeners, Session Recording, Heatmaps	Element Selection, Tool Activation, Confirmation, Navigation
Mouse Movement/Hover	mousemove, mouseover, mouseout events, coordinates, duration	Event Listeners, Session Recording, Heatmaps	User Attention, Exploration, Hover-based Tooltips, Cognitive Load
Drag-and-Drop	Sequence of mousedown, mousemove, mouseup on specific elements	Event Listeners, Session Recording, Custom Event Tracking	Object Movement, Connection Creation, Reordering, Resizing
Keyboard Input (Text)	keydown, keypress, keyup events, character codes, input field ID	Event Listeners, Input Field Monitoring	Text Entry, Labeling, Property Modification
Keyboard Input (Cmd)	keydown, keyup (for shortcuts), command name, context	Event Listeners, Custom Command Logging, Application API Hooks	Tool Activation, Undo/Redo, Global Operations, Mode Switching
Menu/Icon Selection	click on menu item/icon, menu/icon ID, associated command	Event Listeners, Custom Command Logging	Tool Activation, Operation Execution, Feature Access
Gaze Fixation	Eye coordinates, duration of gaze, area of interest (AOI)	Eye-Tracking Software (webcam-based)	User Attention, Focus, Information Seeking, Cognitive Processing
Gaze Path	Sequence of gaze points, saccades, scan path	Eye-Tracking Software (webcam-based)	Visual Exploration Pattern, Reading Behavior, Search Strategy

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

detection of interaction friction. Additionally, a concept to categorize user activity into construction, refinement, and navigation phases is explored to support a structured interpretation of modeling behavior.

2.4.1 Clickstream Analysis

Understanding how users interact with software systems requires the systematic analysis of recorded interaction sequences. Clickstream analysis, originally developed for web usage mining as described by [SCDT00], provides a methodological foundation for extracting behavioral patterns from sequential event data. While the technique emerged from the need to understand website navigation behavior, its principles translate effectively to the analysis of user interactions within graphical modeling environments.

The fundamental idea behind clickstream analysis is treating user interactions as an ordered sequence of discrete events. Each event in this sequence represents a meaningful action, such as creating a model element, modifying a property, or performing an undo operation. By examining these sequences, recurring patterns become visible that would otherwise remain hidden in raw event logs. According to van der Aalst [Aal16], such sequential analysis forms the basis for understanding actual user behavior rather than assumed or designed workflows.

N-Gram Analysis for Behavioral Patterns

A central technique for pattern discovery in event sequences is n-gram analysis. An n-gram represents a contiguous subsequence of n events within the larger interaction stream. For $n = 2$, these subsequences are called bigrams; for $n = 3$, trigrams. Given an event sequence $E = (e_1, e_2, \ldots, e_m)$, the set of all n-grams can be formally defined as:

$$\text{ngrams}(E, n) = \{(e_i, e_{i+1}, \ldots, e_{i+n-1}) \mid 1 \leq i \leq m - n + 1\} \tag{2.1}$$

The frequency distribution of these n-grams reveals which action sequences occur most commonly during modeling sessions. Bigram analysis exposes direct transitions between events, answering questions such as which action typically follows the creation of a class element. Trigram analysis extends this perspective to capture slightly longer behavioral patterns, which can indicate common workflows or habitual interaction sequences.

This approach draws from sequential pattern mining techniques first formalized by R. Agrawal and R. Srikant in [AS95], who introduced algorithms for discovering frequently occurring subsequences in transactional databases. In the context of modeling tool interactions, frequent n-grams may indicate learned workflows, while unusual or unexpected patterns might suggest usability issues or workarounds that users have developed.

The granularity at which events are represented significantly influences the analysis results. A coarse representation using only event types (e.g., `element_create`) provides a high-level behavioral overview but loses contextual detail. A finer representation that combines

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

event type, event kind, and element type (e.g., `element_create:createNode:Type`) enables differentiation between creating different model elements. This distinction matters because creating a class and creating a property likely occur in different behavioral contexts and serve different modeling purposes.

Transition Analysis

Beyond individual n-gram frequencies, the complete transition structure between events provides additional analytical value. A transition matrix T captures the frequency of all observed bigrams in a structured form, where entry T_{ij} represents how often event type e_j immediately followed event type e_i. This matrix representation facilitates the identification of dominant interaction pathways and can reveal asymmetries in user behavior, for instance when users frequently transition from action A to action B but rarely perform the reverse sequence.

Van der Aalst notes that such transition analysis forms the basis for process discovery algorithms that reconstruct behavioral models from event logs. While full process discovery exceeds the scope of simple clickstream analysis, the transition frequencies themselves could provide useful information of how users actually employ the modeling tool [Aal16].

Event Log Export and Process Mining Integration

The eXtensible Event Stream (XES) format, defined in IEEE Standard 1849-2016 [IEE16], has become the de facto standard for event log interchange. An XES file structures event data hierarchically. Logs contain traces (corresponding to individual sessions), and each trace contains a sequence of events with associated attributes such as timestamps, activity names, and domain-specific metadata.

The choice of XES as export format would enable compatibility with a broad ecosystem of process mining tools including ProM, Disco, and the PM4Py library [BvZA19]. These tools offer sophisticated analysis capabilities such as process model discovery, conformance checking, and performance analysis that complement the n-gram based approach. By providing XES export functionality, the interaction data collected from modeling sessions becomes accessible for both lightweight pattern analysis and more comprehensive process mining investigations.

When preparing event data for export, certain low-level events require filtering to produce meaningful analysis results. Mouse clicks and element selections, while important for detailed interaction replay and other analysis methods, introduce noise into behavioral pattern analysis. These events occur with high frequency but carry limited semantic meaning regarding the user's modeling intentions. Excluding such events focuses the analysis on substantive modeling actions, improving the signal-to-noise ratio in discovered patterns.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.4.2 Eye-Tracking

Eye-tracking describes the recording and analysis of eye movements in order to infer where a user is looking during an interaction. In screen-based human-computer interaction settings, the most relevant output is usually the user's point of regard, i.e., the estimated (x, y) position of the gaze on the display. According to Duchowski, this form of data became especially relevant with video-based eye trackers, since they can provide gaze coordinates after calibration and thereby make eye movement data usable for interactive applications and usability studies [Duc17].

For the analysis of modeling tools, eye-tracking is interesting because it adds a visual-attention perspective to ordinary interaction logs. While mouse clicks, keyboard input, and semantic modeling actions show what a user did, gaze data can indicate which regions of the interface or diagram the user visually attended to before or during those actions. This is particularly useful in graphical modeling environments, where users constantly switch between the diagram canvas, tool palettes, properties, labels, and previously created elements. Eye-tracking can therefore help to reveal whether users focus on the expected interface regions, search for modeling elements, repeatedly inspect certain parts of a diagram, or spend substantial time looking at areas that do not directly contribute to the current modeling task.

However, gaze data must be interpreted carefully. Duchowski emphasizes that eye tracking primarily measures overt attention, meaning attention that is aligned with the current fixation point. It cannot prove that the user cognitively processed or fully understood the fixated object, since visual attention and point of regard may diverge [Duc17]. For this reason, eye-tracking data should not be treated as a complete explanation of user behavior on its own. Instead, it is most useful when combined with interaction events, task context, and semantic modeling actions.

In practical analysis, raw gaze recordings are usually too detailed to interpret directly. A short session may already contain thousands of gaze samples, each consisting of timestamped screen coordinates. Therefore, the raw signal first has to be transformed into more meaningful representations. Duchowski describes fixations and saccades as central concepts for eye movement analysis: fixations correspond to relatively stable viewing periods, whereas saccades describe rapid transitions between them [Duc17]. For the purpose of this thesis, the focus is not on a full physiological analysis of eye movements, but on using collected gaze points as an additional data source for visualizing attention during modeling sessions. One common way to make such data interpretable is the generation of fixation or gaze heatmaps, which aggregate many individual gaze samples into a spatial overview of attention distribution.

2.4.3 Fixation Heatmap Visualization

When dealing with large amounts of eye-tracking data, one of the main challenges is finding a way to actually make sense of it all. Raw gaze data can easily contain thousands of coordinate pairs for even a short recording session, which makes direct

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

interpretation essentially impossible. Heatmaps offer a solution to this problem by aggregating gaze points into a single visual representation where color intensity indicates how much attention different screen regions received. This idea is not new—[HNA+11] already describes heat maps as one of the standard approaches to visualizing attention distributions in eye-tracking research.

The basic idea behind generating a heatmap is fairly straightforward. First, the screen area is divided into a grid, and for each cell, the number of gaze points falling within it is counted. This produces a two-dimensional histogram of gaze frequencies. The problem with stopping here, as [Duc17] points out, is that the result looks quite patchy and hard to read since gaze samples are scattered across individual pixels rather than forming coherent regions. To get a smoother and more interpretable image, it is common practice to apply Gaussian filtering to the histogram. Mathematically, this means convolving the raw histogram $H(x, y)$ with a Gaussian kernel:

$$H_{smooth}(x, y) = H(x, y) * G_\sigma(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} H(x - i, y - j) \cdot \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \tag{2.2}$$

The σ parameter here determines how much the data gets blurred. Choosing an appropriate value is somewhat of a balancing act—if σ is too small, the heatmap still looks noisy and fragmented, but if it is too large, distinct attention areas start bleeding into each other and spatial precision suffers. Based on the recommendations by [Boj13], values somewhere between 15 and 30 pixels tend to work reasonably well for typical screen-based eye-tracking setups, though the optimal choice also depends on factors like tracker accuracy and what level of detail the analysis requires.

After smoothing, the values are usually normalized to a $[0, 1]$ range so that consistent color mapping can be applied. The selection of an appropriate color scheme might seem like a minor detail, but it actually matters more than one might initially think. Borland and Taylor [BTI07] showed that the popular rainbow color scale can be misleading because the human visual system does not perceive the color transitions uniformly—some differences appear larger than they actually are in the data, while others get downplayed. Sequential color maps like *hot*, which goes from dark through red and orange to bright yellow, or *viridis* avoid these issues and give a more accurate impression of the actual attention distribution.

For practical analysis, it is often helpful to overlay the heatmap on top of the original stimulus or screenshot. This way, one can immediately see which interface elements or screen regions attracted the most attention without having to mentally map coordinates back and forth. Nielsen and Pernice used this approach extensively in their usability research [NP09] and refer to these composite images as attention maps. To make sure the underlying image remains visible, the heatmap layer is typically rendered semi-transparently with an alpha value around 0.4 to 0.6. In the context of this work, where

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the goal is to understand how users interact with modeling diagrams, being able to see both the diagram elements and the attention patterns simultaneously is quite valuable.

That said, it is worth keeping in mind that heatmaps do have their limitations. Because they aggregate data spatially, all temporal information gets lost in the process. Holmqvist et al. [HNA+11] make the important point that two completely different viewing sequences can result in identical heatmaps as long as they cover the same areas overall. The smoothing step can also be a bit deceptive—it creates the visual impression of continuous attention gradients even in regions where no actual fixations occurred, simply because nearby fixations get blurred outward. For these reasons, heatmaps work best as a first exploratory tool or for communicating high-level findings, but they should probably be complemented with other analysis methods when more detailed questions about viewing behavior need to be answered.

2.4.4 Friction Detection

User friction refers to any interaction pattern that indicates difficulty, confusion, or frustration during system use. Detecting such patterns automatically enables researchers and developers to identify usability problems without relying solely on explicit user feedback or post-hoc questionnaires. As Tullis and Albert [TA13] note, behavioral metrics often reveal usability issues that users themselves fail to articulate or even consciously recognize.

The concept of friction in human-computer interaction draws from broader usability research, where inefficiencies in task completion have long been studied as indicators of interface quality [Nie93]. In the context of graphical modeling tools, friction manifests through specific interaction patterns that deviate from efficient, goal-directed behavior. Four primary friction indicators are particularly relevant for diagram editors: rage clicking, undo/redo bursts, navigation thrashing, and rapid element deletion.

Rage Clicking

Rage clicking is a behavioral metric used in digital experience analytics to describe users rapidly and repeatedly clicking on an interface element. This behavior typically occurs when users perceive the system as unresponsive or when interaction affordances are unclear [Ful22]. While the industry standard (e.g., FullStory, Microsoft Clarity) often formalizes rage clicks as sequences of three or more clicks within a short temporal window (typically 1–2 seconds) [Ful22], this pattern is deeply rooted in established HCI research regarding user frustration and computer anxiety [CLB+03].

The detection of rage clicks requires tracking both the temporal density and spatial clustering of click events. A sliding window approach identifies candidate clusters, which are then validated against a maximum radius threshold to ensure the clicks target approximately the same interface region. The severity of a rage click event scales with the number of clicks in the cluster, as higher click counts suggest more pronounced frustration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Undo/Redo Patterns

Excessive use of undo and redo operations indicates that users are struggling to achieve their intended outcome, either due to unclear action consequences or an exploratory trial-and-error approach forced by insufficient system feedback. Akers et al. [ASJW09] analyzed undo behavior in creative applications and found that bursts of undo operations frequently precede task abandonment, making them valuable predictors of user difficulty.

Particularly diagnostic are oscillating patterns where users alternate between undo and redo operations, suggesting uncertainty about the correct course of action. A pattern such as "undo-undo-redo-undo-redo" reveals that the user cannot confidently commit to a particular system state. GLSP-based editors inherently support undo/redo through their command stack architecture, making these operations straightforward to capture and analyze. Detection involves identifying temporal clusters of undo/redo events and characterizing their sequential patterns to distinguish routine corrections from problematic bursts.

Navigation Thrashing

Navigation thrashing occurs when users make rapid, seemingly erratic changes to the viewport through panning, zooming, or scrolling. This behavior typically emerges when users cannot locate a target element or struggle to maintain orientation within a large or complex diagram. Pirolli and Card [PC99] describe similar patterns in the context of information foraging, where disoriented users engage in inefficient navigation strategies.

In diagram editors, navigation thrashing manifests as multiple viewport changes within a short time window without sustained focus on any particular region. An appropriate detection algorithm will need to track viewport-related events including pan, zoom, and scroll operations, flagging sequences where the frequency exceeds a threshold indicative of purposeful navigation. Measuring the total distance traveled by the viewport during such episodes provides additional insight into the severity of the disorientation.

Rapid Element Deletion

The rapid deletion of multiple diagram elements in quick succession could signal that users are abandoning their current approach, either due to accumulated errors or a fundamental misunderstanding of how to achieve their goal. While individual deletions are routine editing operations, clusters of deletions suggest a more significant problem. Ko, Myers, and Aung [KMA04] identify such "cleaning up" behavior as a common response to cascading errors in programming environments, and similar dynamics apply to graphical modeling.

Detection requires distinguishing between deliberate batch operations, where a user intentionally removes multiple elements as part of their workflow, and frustrated deletions that indicate failure. Temporal clustering serves as the primary discriminator, as deliberate

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

batch operations typically involve selection followed by a single delete action, whereas frustrated deletions appear as repeated individual delete operations in rapid succession.

Composite Friction Scoring

Individual friction indicators gain explanatory power when aggregated into a composite score that characterizes overall session quality. Following the approach of Lewis and Sauro [LS16], who advocate for combining multiple usability metrics into unified scores, the friction indicators can be weighted and combined to produce a single friction score ranging from zero (no detected friction) to one hundred (severe friction throughout the session).

The weighting scheme reflects the relative severity of each indicator type, with rage clicks weighted most heavily due to their strong association with explicit frustration, followed by undo/redo bursts, navigation thrashing, and rapid deletions. The composite score enables comparison across sessions and users while the underlying indicator breakdown supports detailed diagnostic analysis.

2.5 Privacy and Ethical Principles

The envisioned interaction tracking framework for graphical modeling environments will collect detailed data about user behavior, such as mouse movements, keyboard input, and potentially eye-gaze information. To minimize the risk that this data could be linked to identifiable individuals, the design of such a framework must carefully address privacy protection and ethical responsibilities. This section outlines the key legal requirements and ethical principles that guide the responsible development and use of interaction tracking systems.

2.5.1 Regulatory Framework

Within the European Union, the General Data Protection Regulation (GDPR) provides the central legal framework for the collection and processing of personal data. Interaction logs and eye-tracking data qualify as personal data under the GDPR, as they could be associated with individual users and may reveal behavioral or cognitive characteristics [otEU16, KLM20]. Eye-tracking data requires particular attention, as it may fall under the category of sensitive personal data regulated by Article 9 of the GDPR. Prior research has shown that gaze patterns can implicitly disclose biometric identifiers, emotional responses, cognitive load, and health-related information [KLM20, GOH+23]. As a result, systems that incorporate eye-tracking must apply stricter safeguards than those used for conventional interaction data.

2.5.2 Privacy by Design

Privacy by Design (PbD) promotes the integration of privacy protection into systems from the earliest design stages rather than treating it as a later addition [Cav09, Pri26].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This approach is formally embedded in the GDPR under Article 25, which requires data controllers to implement appropriate technical and organisational measures to ensure data protection by default [otEU16].

The core idea of PbD is that privacy should be the default system behavior. For interaction tracking frameworks, this implies that tracking is disabled by default, that users must actively opt in, and that only necessary data is collected. Privacy considerations must be embedded directly into the system architecture, for example through modular data collection, configurable tracking levels, and secure data handling mechanisms.

Operationally, applying PbD means offering clear consent flows and ensuring that the system remains usable even if users decline participation in tracking [Pri]. This helps balance research needs with user autonomy and legal compliance.

2.5.3 Summary of Guiding Principles

The interaction tracking framework described in this work is guided by the following core principles:

- **Lawfulness**: All data processing is based on a valid legal basis, primarily informed consent.
- **Transparency**: Users are clearly informed about data collection, processing, and potential inferences.
- **Data Minimization**: Only data necessary for defined research goals is collected and retained.
- **Purpose Limitation**: Data is used exclusively for stated purposes unless additional consent is obtained.
- **Security**: Technical and organizational measures protect data throughout its lifecycle.
- **User Control**: Participants can access, manage, and withdraw their data and consent.

Together, these principles form the ethical and legal foundation for the design and implementation of the interaction tracking framework. Subsequent sections describe how these principles are translated into concrete architectural and implementation decisions.

2.6 Synopsis

This chapter laid out the conceptual and technical foundations on which the framework proposed in this thesis builds. The opening sections situated interaction tracking in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the broader landscape of User Behavior Analytics, web analytics, and recommendation engines, before narrowing in on the tools and frameworks that have been used to capture user behavior in web applications. The survey contrasted commercial session-replay platforms (Hotjar, FullStory), an open-source product analytics stack (PostHog), research instruments for general web usability (WebTrace, Interaction Lab), and the two prior efforts that explicitly target graphical modeling editors (TOOL and MER). The remaining sections catalogued the user interaction types that are meaningful inside diagram-based modeling environments, the analysis techniques applicable to the resulting event streams, and the legal and ethical framework within which any such tracking system must operate.

Several observations consolidate across these sections and together define the research gap that motivates the remainder of this thesis. First, the commercial and general-purpose open-source tools are mature in feature breadth but have no notion of modeling semantics; they treat the canvas as an unstructured DOM and therefore cannot answer questions about how modelers actually construct a model. Second, the two existing modeling-aware tools each cover only part of the problem: TOOL captures multi-modal data but is closed, monolithic, and hard-wired to two specific modeling languages, while MER captures semantically rich model-change events at the EMF level but ignores the surrounding UI-level behavior – mouse movement, hover dwell, navigation gestures, keyboard shortcuts, gaze fixations – that is decisive for UX research. Third, none of the surveyed tools is designed as a reusable, tool-agnostic library that an arbitrary GLSP-based modeling environment can adopt with low integration effort, and none combines fine-grained UI capture, gaze data, and modeling-semantic awareness under a single, self-hostable architecture.

Closing this gap is the contribution of the framework presented in the remainder of this thesis. The next chapter translates the observations above into concrete functional and non-functional requirements; subsequent chapters describe the resulting design, its reference integration into a GLSP-based modeling tool, and its evaluation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 3

Requirement Analysis

This chapter derives the key requirements for the proposed interaction tracking framework based on the findings of the literature review. The goal is to translate theoretical insights and identified research gaps into concrete system requirements that support empirical UX research while also enabling practical improvements of graphical modeling tools. The requirements are divided into functional and non-functional aspects, followed by specific considerations for the reference integration used in this thesis.

3.1 Functional Requirements

Functional requirements describe what the framework must be capable of doing in order to fulfill its intended purpose. Fundamentally, the system is expected to capture, store, and analyze user interactions in graphical modeling environments in a way that allows researchers and tool developers to better understand modeling behavior.

A fundamental requirement is comprehensive event capture. The framework must record a wide range of user interactions, covering both fine-grained input events and higher-level actions that reflect user intent. On a low level, this includes basic input such as mouse clicks, hover events, and drag-and-drop operations, together with their coordinates and the affected interface elements. Keyboard interactions are equally important for renaming Elements and Attributes. These raw events form the foundation for all subsequent analysis and must be captured reliably and with sufficient temporal precision (timestamps).

Beyond raw input, the framework must also support the identification and logging of semantic actions. These high-level events represent meaningful operations within the modeling process, such as creating, deleting, or modifying model elements and connections. Changes to element positions and adjustments to the current view, for example through panning or zooming, also fall into this category. Capturing such actions allows the system

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to distinguish intentional modeling steps from incidental user input and provides a clearer picture of how users work with a model.

In addition to explicit interactions, the framework should support the collection of implicit behavioral data, in particular eye-tracking information. Using standard webcams, the system should be able to record gaze points and fixation durations on specific elements or areas of interest. This data enables the analysis of visual attention and scanning behavior and can offer valuable insights into cognitive load, search strategies, and moments of uncertainty during modeling tasks.

To make the collected data usable across different tools and studies, a standardized and flexible data model is required. The framework must define a common event schema that can represent diverse interaction types in a consistent way. Core concepts such as the performed action, the target object, the broader activity, and relevant contextual information should be explicitly modeled. Logging should follow an event-based and immutable approach, ensuring that all interactions are recorded as a chronological sequence of events. This makes it possible to reconstruct the modeling process step by step and to analyze causal relationships between actions. Each event should be enriched with contextual metadata, including precise timestamps, a pseudonymized user identifier, the current application state, and relevant environmental information.

Given the potentially large volume of interaction data, efficient data processing and storage are essential. Particularly it should allow the bulk import of historically recorded data, for example from local log files. The storage layer must be scalable and capable of handling data from multiple tools and users in a secure manner. In addition, the system should provide mechanisms to transform raw events into more abstract representations, such as higher-level activities, in order to infer user intent and reduce analytical complexity.

Building on this data foundation, the framework must offer analysis capabilities that support both UX research and tool improvement. Quantitative analyses should include metrics such as the frequency of semantic actions, the time spent on specific tasks or model elements, and indicators of user difficulties. Examples include repeated undo and redo actions, excessive clicking, or patterns suggesting that users struggle to locate certain features. Complementing this, qualitative analysis features are required to better understand modeling workflows. A central component is a replay mechanism that reconstructs the complete modeling process, either in real time or at an accelerated pace. Additional analysis techniques, such as clickstream analysis and heatmaps help visualize user behavior and identify common interaction patterns.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 3.1: Summary of functional requirements.

ID	Requirement
FR1	The framework shall capture low-level user interactions, including mouse, drag-and-drop, and related interaction data.
FR2	The framework shall capture semantically meaningful modeling actions, including the creation, deletion, modification, and movement of model elements and connections.
FR3	The framework shall capture view-manipulation events such as panning and zooming.
FR4	The framework shall support the collection and analysis of eye-tracking data.
FR5	The framework shall provide a standardized and extensible event representation with timestamps, session context, and relevant metadata.
FR6	The framework shall provide persistent storage and support the import and retrieval of recorded interaction sessions.
FR7	The framework shall provide analytical methods for identifying interaction patterns and potential usability issues.
FR8	The framework shall support replay of recorded modeling sessions.
FR9	The framework shall support exporting interaction data for further analysis with external tools.

3.2 Non-Functional Requirements

Non-functional requirements describe the quality attributes of the system and define how the framework should operate rather than what it does. One key aspect is modularity and extensibility. The framework should follow a component-based architecture with loosely coupled modules, making it easier to maintain and extend. Both the data model and the analysis pipeline must be flexible enough to accommodate new interaction types, additional context data, or future extensions. Well-defined APIs are required to allow customization and integration of tool-specific tracking logic, analysis methods, or visualization components.

Another important requirement is tool agnosticism. The framework should be integratable into different graphical modeling environments without being tightly bound to a specific implementation. In particular, it must be compatible with browser-based tools and standard web technologies. Core tracking mechanisms should remain independent of the host tool, minimizing integration effort and reducing the risk of unintended side effects. As a concrete reference, the framework must support integration with the Graphical Language Server Platform (GLSP), demonstrating its applicability in a modern modeling architecture.

Performance and scalability are also critical non-functional concerns. The tracking framework must introduce only minimal overhead and must not noticeably degrade the responsiveness of the host tool.

Finally, the framework must adhere to strict privacy, security, and ethical standards. Data

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

collection should follow the principle of data minimization, ensuring that only information necessary for the intended analysis is recorded. Users must be clearly informed about what data is collected and for what purpose, and explicit consent mechanisms must be provided. Collected data must be protected through appropriate security measures, including encryption and access control. Pseudonymization or anonymization techniques should be applied wherever possible to protect user identities. Transparency regarding data collection, storage, and usage is essential to maintain user trust and to comply with legal and ethical requirements.

Table 3.2: Summary of non-functional requirements.

ID	Requirement
NFR1	The framework shall follow a modular and extensible architecture that allows new interaction types and analysis components to be added.
NFR2	The framework shall remain largely independent of a specific modeling tool and use generic interfaces where possible.
NFR3	The tracking functionality shall introduce only limited performance overhead and shall not noticeably interfere with normal modeling interactions.
NFR4	The framework shall support privacy-conscious and secure data collection, including explicit consent, data minimization, pseudonymization, access control, and deletion of recorded data.
NFR5	The framework shall be compatible with browser-based modeling environments and standard web technologies.

3.3 Reference Integration Specifics

To demonstrate the practical applicability of the proposed framework, a concrete integration with GLSP is required. This integration serves as a reference implementation and shows how the framework can be embedded into an existing modeling infrastructure. It also provides a tangible demonstration of the framework's value for empirical UX research and for identifying opportunities to improve modeling tools based on real user interaction data.

Table 3.3: Requirement for the reference integration.

ID	Requirement
IR1	The framework shall be integrated into a GLSP-based modeling tool as a reference implementation and exercised with real interaction data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Framework Design and Implementation

This chapter presents the architectural design and technical implementation of the interaction tracking framework derived from the above requirements, detailing both the backend infrastructure and frontend instrumentation library. The framework design follows separation of concerns and a plugin-style architecture to keep "tracking code" isolated from the host modeling tool. In practice, this mainly affected how the modules are registered and how events are serialized, deserialized and sent to the backend. The implementation demonstrates how domain-agnostic tracking mechanisms can be adapted to the specific requirements of web-based modeling environments.

4.1 System Architecture Overview

The framework consists of (1) a frontend tracking library in TypeScript that captures real-time interaction events and has a batch export feature, (2) a standalone Eye-Tracking tool via a separate browser window, (3) a Python backend that exposes a REST API for persisting data and analytical tools, and (4) a database schema optimized for storing timestamped event streams. In the reference implementation, the BIGUML Project was extended with the tracking library, in addition WebGazer was used for eye-tracking and FastAPI in combination with Postgres for storage. This architectural separation enables independent scalability, eases maintenance, and supports deployment across heterogeneous modeling environments.

4.1.1 Design Principles

The framework architecture conforms to several key design principles identified as critical for interaction tracking systems in modeling environments:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Tool Agnosticism: The core tracking mechanisms operate independently of specific modeling tool implementations, utilizing standardized web APIs and event interfaces. This design decision ensures broad applicability across different GLSP-based editors and potentially other web-based modeling platforms. In early development integrating the "simple" webcam based eye-tracker into the tracking library itself was approached. This caused multiple issues, which will be explained in a later section. This essentially caused the decoupling of the eye-tracking from the main modeling-editor and import and merge gaze-data later in the backend, which has the side effect of allowing other eye-tracking libraries to be integrated much easier.

Minimal Performance Impact: Frontend instrumentation employs event batching, debouncing, and asynchronous transmission to minimize latency introduced into the host application. In early profiling, the additional overhead was noticeable mainly for high-frequency events (e.g., viewport changes or dragging elements). To keep UI impact low, the implementation throttles these events and batches network requests. A detailed overhead evaluation is reported in Section 5.6.

Extensible Event Model: The event schema supports both predefined interaction types and custom semantic annotations, allowing tool developers to augment basic tracking with domain-specific contextual information relevant to their modeling languages.

Privacy-by-Design: The implementation is designed to minimize personal data collection. Events are linked to a pseudonymous session identifier, and users can enable/disable tracking modalities (e.g., eye-tracking) via explicit consent. GDPR-related considerations (purpose limitation, data minimization, revocation) are discussed in Chapter 5.6. The user has to "opt-in", can access a documentation of exactly what is being captured, and also has the option to delete all of their data.

4.1.2 Backend Infrastructure

The backend subsystem provides RESTful endpoints for data ingestion, persistent storage, and several analytical features. It serves as the central hub for collecting interaction data from instrumented modeling tools. The implementation relies on a straightforward but robust technology stack. The full source code is available on GitHub[1].

FastAPI Framework. FastAPI[2] was selected as web framework due to its native support for asynchronous request handling, automatic request validation through Pydantic models, and built-in OpenAPI documentation generation. The framework automatically produces interactive API documentation accessible via Swagger UI, which proved invaluable during development and integration testing. FastAPI's type hints integration also enabled early detection of schema mismatches between frontend and backend components.

[1] `https://github.com/granigd/InteractionMonitoring`, last accessed 2026-05-11
[2] `https://fastapi.tiangolo.com/`, last accessed 2026-07-22

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

PostgreSQL Database. Rather than implementing a specialized time-series database as initially considered, the implementation uses PostgreSQL for all data storage. This decision was driven by several practical factors: PostgreSQL's JSONB support enables flexible storage of heterogeneous event payloads, its indexing capabilities provide adequate query performance for the expected data volumes, and the operational complexity is significantly lower than managing multiple database systems. The trade-off is that certain analytical queries may be slower than they would be with a columnar database, but this was deemed acceptable for the current scope.

SQLAlchemy ORM. The database interaction layer uses SQLAlchemy with its async support, providing a clean abstraction over the PostgreSQL interface while maintaining the flexibility to execute raw SQL when needed.

Database Schema

The schema of the database is centered on three primary tables that capture different aspects of user interactions. The DDL code is listed in the appendix.

Figure 4.1: Database Schema

Recording Sessions Table. The `recording_sessions` table represents the "header" for a recording. Each row corresponds to one complete tracking session and stores high-level metadata such as which tool produced the data, what model/editor was open, who

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

recorded it (if provided), and when the session started and ended. It also stores summary values like total duration and number of events, plus an `extra_data` field for optional, tool-specific metadata.

The schema is intentionally generic, using field names like `tool_id` and `editor_type` rather than tool-specific terminology. This design enables the same backend to serve multiple GLSP-based tools without modification.

Interaction Events Table. The `interaction_events` table stores the actual interaction stream. Each row is one interaction event that belongs to a session (via the session foreign key) and is ordered using timestamps and a sequence number. It records a coarse category (`event_type`) and optional subtype (`event_kind`), while the complete event payload is preserved in `data` (JSONB) for replay and later analysis. Frequently queried fields such as element identifiers and coordinates are duplicated into dedicated columns to make filtering and indexing efficient.

Gaze Points Table. The `gaze_points` table stores raw eye-tracking samples for a session. Because gaze data can be high-frequency, it is kept separate from interaction events and optimized for simple inserts and time-based queries. Each row contains the sample timestamp and screen coordinates (optionally normalized), quality indicators such as confidence, and a `raw_data` JSONB field to retain tracker-specific information. Fields like `fixation_id` and `is_fixation` are reserved for post-processing steps (e.g., fixation detection) that can group raw samples into more meaningful attention units.

API Design

The REST API follows resource-oriented design principles, organizing endpoints into logical groups that reflect the domain model of interaction tracking. The complete API documentation is automatically generated through FastAPI's OpenAPI integration and accessible via Swagger UI, which proved invaluable during development and client integration. Figure 4.2 illustrates the complete endpoint structure.

The session management endpoints form the core of the data ingestion layer. The primary endpoint accepts complete recording sessions as JSON payloads containing session metadata and an array of interaction events. This design decision to upload entire sessions rather than streaming individual events simplifies the client implementation and ensures data consistency, as sessions are stored atomically. The endpoint validates incoming data through Pydantic models that enforce type constraints and required fields, returning detailed error responses when validation fails. Upon successful upload, the backend extracts and denormalizes key information from event payloads to enable efficient querying. This extraction logic handles common GLSP action types such as element creation, selection, bounds changes, and label edits, while providing fallback handling for unrecognized actions that still attempts to extract element information from standard field names.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

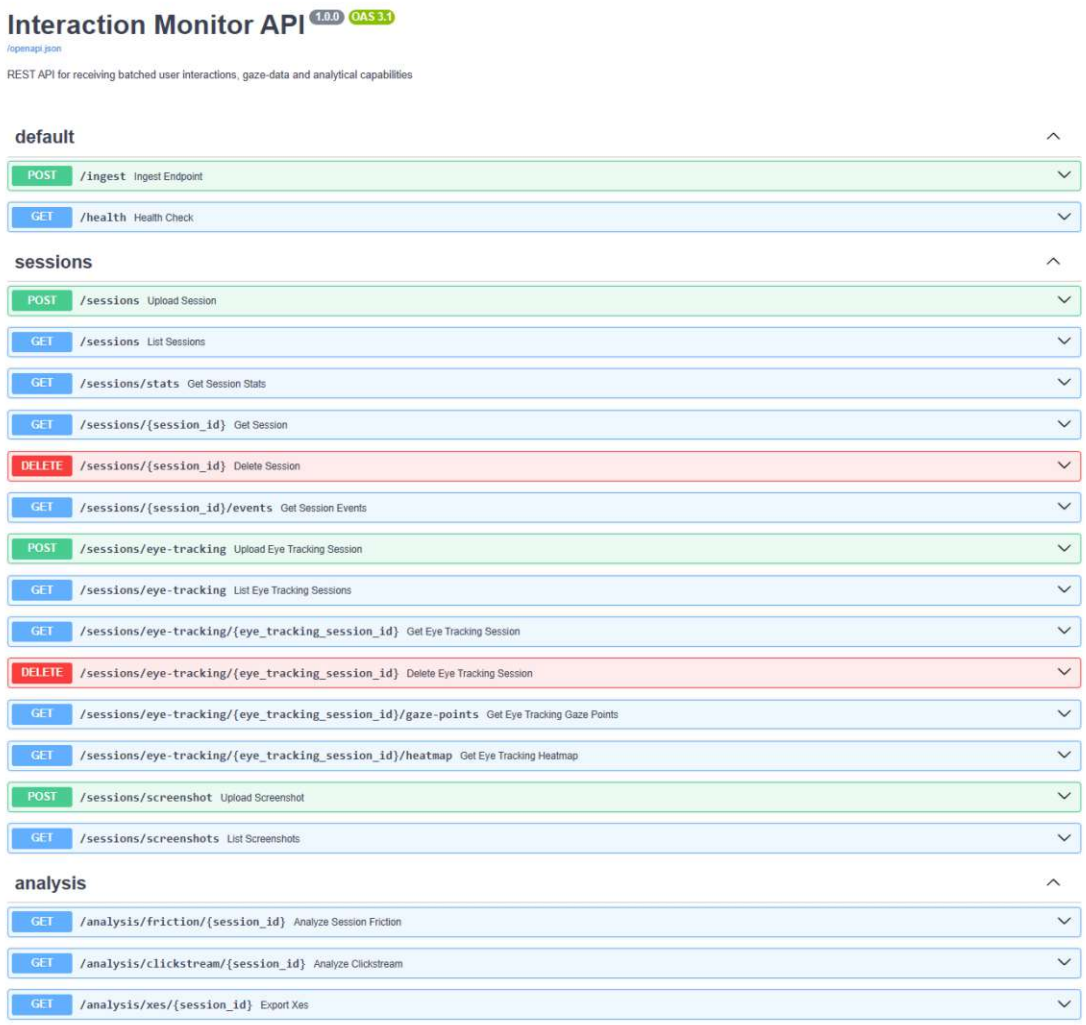

Figure 4.2: Swagger documentation of the backend REST-API

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Session retrieval endpoints support pagination and filtering by multiple criteria including tool identifier, editor type, user name, and date ranges. This flexibility enables researchers to query specific subsets of data for focused analysis. A dedicated statistics endpoint aggregates information across all sessions, providing counts by tool, editor type, and user, as well as average session duration and event type distributions. Individual session retrieval returns complete event histories with optional filtering by event type, element identifier, or sequence number ranges.

Eye-tracking data ingestion follows a similar pattern through dedicated endpoints that accept gaze point arrays with associated metadata. The API supports linking eye-tracking sessions to interaction recording sessions through session identifiers, enabling temporal correlation during analysis. Gaze point retrieval endpoints support pagination to handle the potentially large volumes of eye-tracking data, with options to filter by sequence number ranges.

The heatmap generation endpoint deserves particular mention as it transforms raw gaze data into visual representations on demand. Rather than pre-computing heatmaps during upload, the API generates them dynamically based on query parameters including Gaussian smoothing sigma, colormap selection, and overlay transparency. When a screenshot exists for the associated session, the endpoint automatically overlays the heatmap on the screenshot, providing immediate visual context for attention analysis.

For process mining integration, the API provides XES export functionality that transforms session event logs into the IEEE 1849-2016 standard format. The exporter filters out low-level noise events such as raw mouse clicks and element selections to focus on semantically meaningful actions. Event names in the exported XES files use full granularity combining event type, event kind, and element type to enable differentiated analysis across different modeling operations.

All endpoints require API key authentication to prevent unauthorized access, with the key validated through a dependency injection pattern that keeps authentication logic separate from business logic. Session deletion endpoints support GDPR compliance by allowing complete removal of user data including all associated events and gaze points through cascading database deletes.

4.2 Implemented Analytical Methods

Building upon the theoretical foundations established in Chapter 2, the framework implements three complementary analytical methods that transform raw interaction data into digestible information. These methods address different aspects of user behavior analysis: sequential pattern discovery through clickstream analysis, usability problem detection through friction analysis, and visual attention mapping through heatmap generation. Each method was designed to operate on the standardized event data model stored in the backend database, enabling consistent analysis across different sessions and tools.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2.1 Clickstream Analysis Implementation

The clickstream analysis module translates the theoretical concepts of n-gram analysis and transition frequency measurement into a practical tool for behavioral pattern discovery. The implementation is contained in `clickstream_analyzer.py` and relies on Python's standard library `collections.Counter` for efficient frequency counting and `xml.etree.ElementTree` for XES export generation.

A fundamental design decision concerns event representation granularity. The ClickstreamAnalyzer class accepts a granularity parameter that controls how event labels are constructed:

Listing 4.1: Event label construction with configurable granularity

```
def _get_event_label(self, event: dict) -> str:
    """Build event label based on granularity setting."""
    event_type = event.get('event_type', 'UNKNOWN')

    if self.granularity == 'type':
        return event_type  # e.g., 'element_create'

    event_kind = event.get('event_kind') or ''
    if self.granularity == 'kind':
        if event_kind:
            return f"{event_type}:{event_kind}"  # e.g., '
                element_create:createNode'
        return event_type

    # granularity == 'full'
    element_type = event.get('element_type') or ''
    parts = [event_type]
    if event_kind:
        parts.append(event_kind)
    if element_type:
        parts.append(self._clean_element_type(element_type))
    return ':'.join(parts)  # e.g., 'element_create:createNode:Class'
```

This configurable granularity allows researchers to tune the analysis to their specific questions – coarse granularity reveals broad workflow patterns while fine granularity distinguishes between operations on different model elements. The `_clean_element_type` helper removes namespace prefixes (e.g., converting `CLASS__Property` to `Property`) for cleaner output.

Before computing n-grams, the analyzer filters the event stream to exclude low-level interactions that introduce noise without contributing semantic meaning. Mouse clicks and element selections occur with such high frequency that they dominate pattern statistics without reflecting intentional modeling decisions:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Listing 4.2: Event filtering and sequence extraction

```
EXCLUDED_EVENTS = {'mouse_click', 'element_select'}

def extract_sequence(self, events: List[dict]) -> List[str]:
    """Extract ordered sequence of event labels (excluding noise)."""
    return [self._get_event_label(e) for e in events
            if self._should_include(e)]
```

The n-gram computation follows the formal definition presented in Section 2.4.1. Given an event sequence $E = (e_1, e_2, \ldots, e_m)$, the implementation generates all contiguous subsequences of length n:

Listing 4.3: N-gram computation

```
def compute_ngrams(self, sequence: List[str], n: int) -> List[tuple]:
    """Compute n-grams from a sequence."""
    if len(sequence) < n:
        return []
    return [tuple(sequence[i:i+n]) for i in range(len(sequence) - n +
        1)]
```

The `Counter` class from Python's collections module efficiently tallies n-gram frequencies, enabling identification of the most common bigrams and trigrams. The implementation also constructs a transition matrix as a nested dictionary structure where `transitions[a][b]` contains the count of how often event b immediately followed event a.

For process mining integration, the `XESExporter` class transforms sessions into the IEEE 1849-2016 XES format using ElementTree for XML construction and `xml.dom.minidom` for pretty-printing. The exporter registers standard XES extensions (Concept, Time, Lifecycle) and maps session metadata to trace-level attributes while individual events become XES events with semantic naming:

Listing 4.4: XES event construction

```
# Build event name: event_type:event_kind:element_type
name_parts = [event_type]
if event_kind:
    name_parts.append(event_kind)
if element_type:
    name_parts.append(self._clean_element_type(element_type))
event_name = ':'.join(name_parts)

self._add_string_attr(event_elem, 'concept:name', event_name)
self._add_date_attr(event_elem, 'time:timestamp', ts_str)
self._add_string_attr(event_elem, 'lifecycle:transition', 'complete')
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The exported XES files are compatible with standard process-mining tools including ProM[3], Disco[4], and PM4Py [BvZA19], enabling researchers to apply process discovery and conformance-checking algorithms to modeling-session data.

4.2.2 Friction Detection Implementation

The friction detection module, implemented in `friction_detector.py`, operationalizes the theoretical indicators described in Chapter 2 into automated detection algorithms. The implementation uses a sliding window approach for temporal clustering and computes Euclidean distances for spatial analysis using Python's `math` module. The `FrictionDetector` class accepts a configuration object that allows researchers to tune detection thresholds.

Each friction type requires specialized detection logic. Rage click detection identifies sequences of rapid, spatially concentrated clicks suggesting user frustration. The algorithm uses helper functions for geometric calculations:

Listing 4.5: Geometric helper functions for rage click detection

```
def calculate_distance(x1: float, y1: float, x2: float, y2: float) ->
    float:
    """Calculate Euclidean distance between two points."""
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

def calculate_centroid(points: List[Tuple[float, float]]) -> Tuple[
   float, float]:
    """Calculate centroid of a set of points."""
    if not points:
        return (0.0, 0.0)
    x_sum = sum(p[0] for p in points)
    y_sum = sum(p[1] for p in points)
    return (x_sum / len(points), y_sum / len(points))
```

The core detection algorithm iterates through click events using a sliding window. For each potential cluster starting point, it collects subsequent clicks within the configured time window (default: 1500ms) and verifies spatial proximity to the cluster centroid (default maximum radius: 50 pixels):

Listing 4.6: Rage click sliding window detection (simplified)

```
for i, click in enumerate(click_events):
    if i in used_indices:
        continue

    cluster = [click]
    # Find clicks within time window
```

[3] https://www.promtools.org/
[4] https://fluxicon.com/disco/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    for j in range(i + 1, len(click_events)):
        other = click_events[j]
        time_diff = (other['timestamp'] - click['timestamp']).
            total_seconds() * 1000

        if time_diff > self.config.rage_click_time_window_ms:
            break

        # Check spatial proximity to cluster centroid
        points = [(c['x'], c['y']) for c in cluster]
        centroid = calculate_centroid(points)
        distance = calculate_distance(centroid[0], centroid[1], other
            ['x'], other['y'])

        if distance <= self.config.rage_click_max_radius_px:
            cluster.append(other)

    # Flag as rage click if minimum threshold met
    if len(cluster) >= self.config.rage_click_min_clicks:
        # Create RageClickEvent with severity based on click count
        ...
```

Undo/redo burst detection targets patterns of excessive correction operations. The implementation identifies undo and redo events by checking both the `event_kind` field and the nested `data.kind` field against known identifiers (`undo`, `UndoAction`, `redo`, `RedoAction`). Beyond counting operations, the detector constructs a pattern string that captures the sequence structure:

Listing 4.7: Undo/redo pattern analysis

```
# Create pattern string: 'U' for undo, 'R' for redo
pattern = ''.join('U' if c['type'] == 'undo' else 'R' for c in
    cluster)

# Higher severity for oscillating patterns (back-and-forth)
has_oscillation = 'UR' in pattern or 'RU' in pattern
severity_count = len(cluster)
if has_oscillation:
    severity_count += 2  # Boost severity for confusion indicators
```

The pattern string enables qualitative interpretation. A pattern like "UUUUU" indicates pure undo (reverting multiple changes), while "URURUR" reveals oscillation suggesting the user cannot decide on the correct state. The implementation includes an `_interpret_pattern` method that generates human-readable descriptions of detected patterns.

Navigation thrashing detection tracks viewport-related events including `setViewport`, `center`, `fit`, `scroll`, and `zoom` operations. When multiple such events occur within

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the configured time window (default: 5 seconds), the detector flags this as a thrashing episode and calculates the total viewport distance traveled by summing Euclidean distances between consecutive viewport positions.

Rapid deletion detection uses similar sliding window logic to identify clusters of delete-element operations within the configured time window (default: 3 seconds). The implementation tracks the types of deleted elements to help identify whether particular model element categories are associated with higher deletion rates.

The composite friction score synthesizes all detected patterns using a weighted formula. Each friction type contributes a component score scaled from 0 to 100 based on event counts and severity levels:

Listing 4.8: Composite friction score calculation

```
# Weighted combination of component scores
weights = {
    'rage': 0.35,       # Highest weight - strong frustration signal
    'undo_redo': 0.30,
    'navigation': 0.20,
    'deletion': 0.15,
}

overall = (
    rage_score * weights['rage'] +
    undo_score * weights['undo_redo'] +
    nav_score * weights['navigation'] +
    deletion_score * weights['deletion']
)
```

The weights were determined through iterative refinement during development, with rage clicks weighted most heavily due to their strong association with explicit frustration. The final score maps to interpretive categories: 0–20 indicates minimal friction, 21–40 low friction, 41–60 moderate friction, 61–80 high friction, and 81–100 critical friction.

The friction analysis response includes automatically generated recommendations based on detected patterns. For example, when rage clicks are detected, the system suggests investigating the specific screen positions and recommends improving UI responsiveness or element visibility in those areas.

4.2.3 Heatmap Visualization Implementation

The heatmap generation implementation transforms raw gaze point collections into visual representations of attention distribution. The implementation leverages NumPy for efficient array operations, SciPy's `ndimage` module for Gaussian filtering, Matplotlib for rendering, and Pillow (PIL) for image handling. The core algorithm is implemented both as a standalone script (`heatmap.py`) and integrated into the sessions API endpoint for on-demand generation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The generation pipeline begins by extracting coordinate arrays from the gaze point collection and constructing a two-dimensional histogram using NumPy's `histogram2d` function:

Listing 4.9: Heatmap histogram construction

```
import numpy as np
from scipy.ndimage import gaussian_filter

# Clip coordinates to image bounds
x_clipped = np.clip(x_coords, 0, width - 1)
y_clipped = np.clip(y_coords, 0, height - 1)

# Create 2D histogram with pixel-level bins
heatmap, xedges, yedges = np.histogram2d(
    y_clipped,  # rows (y-axis)
    x_clipped,  # cols (x-axis)
    bins=[height, width],
    range=[[0, height], [0, width]],
)
```

The raw histogram appears fragmented due to the discrete nature of individual gaze samples. Gaussian smoothing transforms this into a visually interpretable heatmap by convolving the data with a two-dimensional Gaussian kernel. SciPy's `gaussian_filter` function efficiently implements this convolution:

Listing 4.10: Gaussian smoothing and normalization

```
# Apply Gaussian smoothing (sigma controls blur radius)
heatmap_smooth = gaussian_filter(heatmap, sigma=18)

# Normalize to [0, 1] range for consistent color mapping
max_val = heatmap_smooth.max()
if max_val > 0:
    heatmap_norm = heatmap_smooth / max_val
else:
    heatmap_norm = heatmap_smooth
```

The sigma parameter controls the degree of smoothing. Following recommendations from eye-tracking literature, the default value is set to 18 pixels, which balances visual coherence against spatial precision for typical screen-based setups. The API exposes this parameter (configurable from 1 to 100) for researchers with different accuracy requirements.

Color mapping uses Matplotlib's `imshow` function with a configurable colormap. The default "hot" colormap progresses from dark through red and orange to bright yellow, providing intuitive interpretation where warmer colors indicate higher attention density. This follows visualization research recommendations to avoid perceptual distortions associated with rainbow color scales:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Listing 4.11: Heatmap rendering with optional screenshot overlay

```
import matplotlib.pyplot as plt
from PIL import Image

if screenshot_path and screenshot_path.exists():
    # Overlay mode: render heatmap on top of screenshot
    img = Image.open(screenshot_path)
    fig, ax = plt.subplots(figsize=(img_width/dpi, img_height/dpi),
        dpi=dpi)
    ax.imshow(img)
    ax.imshow(heatmap_norm, origin="upper", alpha=0.45, cmap="hot")
else:
    # Plain heatmap mode
    fig, ax = plt.subplots(figsize=(width/dpi, height/dpi), dpi=dpi)
    ax.imshow(heatmap_norm, origin="upper", cmap="hot")

ax.axis("off")
plt.tight_layout(pad=0)
```

The overlay capability is particularly valuable for practical analysis. When a screenshot exists for the session, the implementation renders the semi-transparent heatmap (default alpha: 0.45) on top of the screenshot image, immediately revealing which interface elements received the most visual attention. The implementation handles dimension mismatches between stored screen dimensions and actual screenshot size through proportional coordinate scaling.

The API endpoint returns the generated image as a streaming PNG response using Matplotlib's `savefig` to write to an in-memory buffer. Response headers include metadata about the session identifier, gaze point count, and whether a screenshot overlay was applied. This streaming approach avoids storing generated heatmaps, keeping storage requirements minimal while allowing unlimited regeneration with different parameters.

The implementation documentation explicitly acknowledges the inherent limitations of spatial aggregation. Heatmaps collapse all temporal information, meaning two entirely different viewing sequences produce identical visualizations if they cover the same areas overall. The Gaussian smoothing can also suggest continuous attention in regions between actual fixations where no gaze data exists. These limitations are communicated through API documentation to ensure appropriate interpretation of generated visualizations.

4.3 Frontend Tracking Library

The client-side tracking library provides the instrumentation layer that captures user interactions within the modeling tool and transmits them to the backend. The library is implemented in TypeScript and designed as a VSCode extension module that integrates with the GLSP client architecture.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.3.1 Architecture Overview

The tracking library follows a service-oriented architecture with clear separation between event capture, session management, and data export.

The central component is the `InteractionTracker` service, which coordinates all tracking activities. This injectable service maintains an array of captured interaction events, the current session metadata, and a collection of disposable resources for proper cleanup. The service implements viewport change debouncing with a configurable timeout (defaulting to 100 milliseconds) to prevent excessive event generation during continuous pan and zoom operations. When a session starts, the tracker automatically detects the diagram type from the active file, extracts the model file path, and captures workspace context including user information and tool version. Part of the implementation is provided in Appendix B.1.

The library defines a comprehensive enumeration of interaction event types that categorize all trackable user actions. These types span diagram interactions (selection, creation, deletion, movement, resizing, and editing of elements), view changes (viewport, zoom, and scroll modifications), property modifications, file operations (open, save, close, and text edits), and session lifecycle events. This classification scheme distinguishes between low-level input events and semantic modeling actions, enabling both fine-grained behavioral analysis and higher-level workflow reconstruction. The complete event type enumeration is shown in Appendix B.2.

4.3.2 GLSP Action Interception

The most critical capability of the tracking library is its interception of GLSP protocol actions. Rather than relying solely on DOM events, the library hooks into the GLSP action dispatch mechanism to capture semantically meaningful operations.

The action listener integration registers handlers for both client-originated actions (from the diagram webview) and server responses. This bidirectional listening ensures that both user-initiated operations and server-confirmed model changes are captured. The registration process creates disposable resources that are properly cleaned up when tracking stops, preventing memory leaks during extended sessions. The listener registration code is detailed in Appendix B.3.

The action processing logic translates intercepted GLSP actions into standardized interaction events. The processor examines the `kind` property of each action to determine appropriate handling. Node creation actions capture the element type identifier, container reference, and creation location. Edge creation actions record the element type along with source and target element identifiers. Selection actions track both selected and deselected element collections. Label edit actions capture the label identifier and new text content. Bounds change actions record the new position and size information. This translation layer abstracts tool-specific action formats into a consistent event schema suitable for analysis. The processing implementation appears in Appendix B.4.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.3.3 Viewport Tracking with Debouncing

Viewport changes (pan and zoom operations) occur at high frequency during user interaction. Without throttling, these events would dominate the event stream and introduce significant performance overhead. The library implements debouncing to capture the final viewport state after continuous manipulation.

The debouncing mechanism works by storing only the most recent viewport event data and resetting a timeout on each new event. When the user stops manipulating the viewport, the timeout expires and the final state is recorded. This approach reduces viewport events from potentially hundreds per manipulation to a single event representing the final state, dramatically reducing storage requirements while preserving the information needed for analysis. The debounced tracking implementation is provided in Appendix B.5.

Additionally, dedicated handlers on the GLSP client side capture viewport state changes and dispatch them to the VSCode extension. The `ViewportTrackingHandler` implements the `IActionHandler` interface and listens specifically for `SetViewportAction` messages. When such actions occur, the handler extracts the scroll position and zoom level from the new viewport state and forwards this information to the tracking system. Importantly, the handler returns undefined to ensure the action continues propagating through the GLSP action chain without interruption. This handler implementation is shown in Appendix B.6.

4.3.4 Element Bounds Tracking

Element move and resize operations, particularly for nested elements within containers, required special handling. Initial testing revealed that some bounds changes were not being captured by the standard action listener when elements were moved within their parent containers. A dedicated `ElementBoundsTrackingHandler` was implemented to address this gap.

The handler maintains a map of pending bounds changes keyed by element identifier, storing both position coordinates (x, y) and size dimensions (width, height). When a `ChangeBoundsOperation` action is intercepted, the handler iterates through all bounds specifications in the action, recording each element's new position and size. Like the viewport handler, this handler returns undefined to allow the action to propagate normally. The bounds tracking handler code is detailed in Appendix B.7.

4.3.5 Data Export

When a tracking session ends, the library exports the collected data in both JSON and CSV formats. The export process first converts all internal millisecond timestamps to ISO 8601 format for interoperability with analysis tools. The session metadata and event array are combined into a single JSON document and written to a file named with the session identifier. Simultaneously, a CSV version is generated for compatibility with spreadsheet applications and statistical analysis tools.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The JSON format preserves the full event structure including nested data objects, enabling complete replay capability and detailed programmatic analysis. The CSV format flattens the event structure into tabular form, facilitating import into statistical analysis tools or serving as input for the client-side replay feature. Both files are written asynchronously to avoid blocking the user interface. The export implementation is provided in Appendix B.8.

4.3.6 Diagram Type Detection

To properly categorize sessions, the library implements automatic diagram type detection based on the active file. The detection algorithm examines the active tab in VSCode, extracts the file URI, and applies a regular expression to match against known UML diagram file extensions. The pattern recognizes activity, class, communication, deployment, information flow, package, state machine, use case, and sequence diagram types. When no match is found, the diagram type defaults to "unknown" to ensure sessions are still recorded even for unrecognized diagram types. The detection implementation appears in Appendix B.9.

4.4 Eye-Tracking Integration

Integrating eye-tracking into the browser-based modeling environment presented unique challenges not encountered with conventional input tracking. The implementation evolved through several iterations as initial approaches proved impractical.

4.4.1 Design Decision: Standalone Application

Early development attempted to embed the webcam-based eye-tracker (WebGazer.js) directly into the GLSP client running within VSCode's webview. This approach encountered multiple obstacles:

- **Security Restrictions.** VSCode webviews operate under a restrictive Content Security Policy that complicated camera access and the loading of WebGazer's TensorFlow models.
- **Performance Impact.** Running real-time gaze prediction within the same process as the diagram renderer caused noticeable lag during intensive modeling operations.
- **Calibration Complexity.** WebGazer calibration requires clicking on reference points across the screen, but the diagram canvas only occupies a portion of the window, limiting effective calibration coverage.

These challenges led to an architectural decision to implement eye-tracking as a standalone browser application running in a separate window. This approach offers several advantages:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Full-screen calibration coverage.
- Complete isolation of eye-tracking processing from the modeling tool.
- Standard browser environment with unrestricted API access.
- Flexibility to switch eye-tracking technologies without modifying the main application.

4.4.2 Standalone Eye-Tracking Implementation

The standalone eye-tracking application is implemented as a single HTML file that can be opened in any modern browser. The application structure consists of a status display section showing the current tracking state and accumulated gaze point count, control buttons for starting tracking, initiating calibration, and exporting collected data, and a calibration overlay containing a 9-point grid for accuracy calibration. The complete HTML structure is provided in Appendix C.1.

The application integrates WebGazer.js for webcam-based gaze estimation. The initialization process configures WebGazer to use ridge regression for gaze prediction and the TFFacemesh tracker for face detection. A gaze listener callback processes incoming gaze data, filtering out invalid readings where coordinates are missing. The system displays the webcam video feed and prediction visualization during operation to provide users with feedback about tracking quality. The WebGazer initialization code is detailed in Appendix C.2.

The gaze data handler creates structured gaze point objects containing x and y screen coordinates along with millisecond timestamps. Each valid gaze point is appended to the collection array and optionally displayed as visual feedback on screen, allowing users to verify that their gaze is being tracked correctly.

The application provides a 9-point calibration interface that covers the entire screen. The calibration implementation enforces sequential point clicking to ensure users follow the proper calibration procedure. When a calibration point is clicked, the handler verifies the point matches the expected sequence, calculates the screen coordinates of the point center, and records multiple samples (five per point) to improve accuracy. Points are visually marked as completed, and calibration automatically concludes after all nine points are processed. The calibration handling code appears in Appendix C.3.

Collected gaze data can be exported as JSON for later analysis or uploaded to the backend. The export function constructs a metadata object containing the export timestamp, total point count, and session duration calculated from the first and last gaze timestamps. This metadata accompanies the complete gaze point array in the exported JSON document. The export creates a downloadable blob file with an automatically generated filename incorporating the export timestamp. The export implementation is shown in Appendix C.4.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.4.3 VSCode Integration

The modeling tool includes a command to launch the standalone eye-tracking application. The command handler constructs the path to the standalone HTML file within the extension's package structure, verifies the file exists, and opens it in the system's default browser using VSCode's external URI opening capability. This approach allows the eye-tracking application to run with full browser permissions while remaining easily accessible from within the modeling environment. The launch command implementation is provided in Appendix C.5.

4.4.4 Limitations and Future Work

The current eye-tracking implementation has several limitations that should be acknowledged:

- **Manual Data Correlation.** Gaze data collected in the standalone application must be manually exported and uploaded to the backend. Automatic synchronization with interaction events is planned for future development.
- **Accuracy Constraints.** WebGazer's accuracy (typically 100-150 pixel error) is sufficient for region-level attention analysis but inadequate for precise element-level fixation mapping in diagrams with small UI components.
- **Coordinate System Translation.** The standalone application captures screen coordinates, which must be transformed to diagram canvas coordinates accounting for viewport pan and zoom. This transformation is not yet implemented.

4.5 Interaction Replay

A capability that emerged as essential during development was the ability to replay recorded sessions and reconstructing the modeling process from captured events. This feature proved valuable for validating tracking completeness and enables future applications in user behavior analysis and modeling process visualization.

4.5.1 Replay Architecture

The replay system consists of three main components:

1. **Event Parser.** Reads and parses the recorded CSV session log and filters out negligible events like (repeated) mouse-clicks. Events are ordered chronologically.
2. **Action Synthesizer.** Translates logged interaction events back into GLSP protocol actions.
3. **ID Mapping Engine.** Maintains correspondence between element IDs in the recording and newly created elements during replay.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.5.2 The ID Mapping Challenge

The most significant technical challenge in the replay implementation was handling the element identifier mapping. GLSP assigns unique UUIDs to model elements at creation time, meaning that the same conceptual element has different IDs in the original session and the replay session.

Consider a recorded sequence where the user creates a class named "Customer" and then connects it to another class with an association. The recorded events reference the original UUID of "Customer". During replay, creating a new class generates a different UUID, but subsequent events (like edge creation) still reference the original ID.

The implementation uses a multi-phase algorithm to address this challenge.

In the first phase (Pre-Analysis), before replay begins, the entire event sequence is scanned to associate creation operations with their corresponding element IDs. The algorithm exploits the temporal ordering: element IDs first appear in selection or property modification events shortly after creation. The pre-analysis identifies all `createNode` events and records their positions and element types. It then scans the entire event sequence to find the first occurrence of each element ID in subsequent events (selections, edge connections, property changes). Finally, it assigns each discovered ID to the temporally nearest preceding creation event of a compatible type. This mapping is stored as a special property on the creation event for use during replay. The complete pre-mapping algorithm is provided in Appendix D.1.

In the second phase (Runtime ID Capture), during replay, after each creation action, the system listens for the GLSP server's model update response to capture the newly assigned ID. The listener registers for `updateModel` actions and extracts all element information from the new model root. When a creation is pending, the system searches for newly appeared elements matching the expected type that were not previously known. The first matching element's ID is captured and associated with the original ID through the mapping table. The runtime capture implementation is shown in Appendix D.2.

In the third phase (ID Translation), subsequent actions undergo ID translation before dispatch. When replaying edge creation events, the system looks up both source and target element IDs in the mapping table. If mappings exist for both endpoints, the action is dispatched with the translated IDs. If either mapping is missing (indicating the referenced element was not successfully created during replay), the edge creation is skipped with a warning. The translation logic for edge creation appears in Appendix D.3.

4.5.3 Container Hierarchy Handling

UML diagrams have hierarchical containment relationships. Activity nodes must be created inside an Activity, states inside a Region, and so on. The replay system tracks container context through element selections.

The container tracking mechanism maintains references to the most recently selected elements of each container type (Activity, Class, Interface, Region, etc.). When creating

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

elements that require containers, the system consults these references to determine the appropriate parent. The implementation defines sets of element types that belong to each container category and provides a lookup function that returns the appropriate container ID based on the element being created. The container tracking implementation is detailed in Appendix D.4.

4.5.4 Playback Control

The replay system supports variable speed playback. The speed parameter controls the temporal compression of the replay, where a speed of 1 reproduces original timing, higher values accelerate playback, and a speed of 0 triggers instant replay without delays.

During replay, the system iterates through events sequentially, reporting progress to the user interface. After dispatching each event, the system calculates the appropriate delay based on the time difference to the next event divided by the speed factor. A minimum delay (100 milliseconds) ensures the UI remains responsive even at high speeds. The replay can be aborted at any time through a cancellation signal, allowing users to interrupt long replay sessions. The playback control implementation is provided in Appendix D.5.

During testing multiple instances occurred where choosing a higher replay-speed sometimes resulted in an incomplete diagram. This happens when the GLSP Server takes too long to respond, and the replay feature already issues the next request.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

bigUML Integration

This chapter details the integration of the tracking framework into bigUML [MB23], a GLSP-based UML modeling tool developed according to the reference architecture proposed by Metin and Bork [MB25]. The integration demonstrates the framework's practical applicability while revealing tool-specific adaptation requirements. The code can be found on a GitHub fork of the original bigUML repository[1].

5.1 bigUML Overview

bigUML is a comprehensive UML modeling tool built on the Graphical Language Server Platform (GLSP) [MB23]. It supports multiple UML diagram types including Activity, Class, Communication, Deployment, Information Flow, Package, Sequence, State Machine, and Use Case diagrams. The tool runs as a VSCode extension, leveraging VSCode's webview infrastructure for the graphical editor while communicating with a Java-based GLSP server for model management.

The architecture follows the GLSP reference architecture [BLO23], consisting of:

- **GLSP Client (TypeScript/Sprotty).** Renders the diagram canvas and handles user interactions, translating them into GLSP protocol actions.
- **GLSP Server (Java/TypeScript).** Processes actions, manages the UML metamodel, and persists changes to EMF-based storage.
- **VSCode Extension.** Hosts the client in a webview, manages communication between client and server, and provides IDE integration features.

[1] `https://github.com/granigd/bigUML`, last accessed 2026-05-11

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

This architecture influenced the integration strategy, as tracking instrumentation needed to span multiple architectural layers.

5.2 Integration Approach

The tracking library was integrated as a new package within bigUML's monorepo structure. The package declares dependencies on GLSP protocol and client libraries at version 2.2.0, ensuring compatibility with bigUML's existing GLSP infrastructure. The package configuration is shown in Appendix E.1.

5.2.1 Module Registration

The tracking module is registered in bigUML's dependency injection container using the InversifyJS binding syntax. The `InteractionTracker` and `InteractionReplayService` are bound as singletons to ensure consistent state management across the extension's lifecycle. The `EyeTrackingProvider` webview component is similarly registered for the user interface panel. The module registration code is provided in Appendix E.2.

5.2.2 GLSP Client Module

A dedicated container module registers the tracking handlers in the GLSP client. The module binds the `ViewportTrackingHandler` and `ElementBoundsTrackingHandler` as singletons and registers them with the GLSP type system as `IActionHandler` implementations. This registration ensures the handlers receive all relevant GLSP actions during diagram interaction. The GLSP client module configuration appears in Appendix E.3.

5.3 User Interface Integration

A dedicated sidebar panel provides the user interface for controlling tracking sessions. The `EyeTrackingProvider` extends bigUML's React webview infrastructure and injects the `InteractionTracker` service. The provider's connection handler registers listeners for tracking-related actions from the webview, including session start, session stop, and server upload requests. Each action type is routed to its corresponding handler method. The provider implementation is detailed in Appendix E.4.

The panel provides buttons for starting and stopping tracking sessions, launching the standalone eye-tracking application, and uploading recorded sessions to the backend server.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.4 Server Upload Integration

The integration includes functionality to upload recorded sessions directly to the monitoring backend. The upload handler presents a file selection dialog filtered for JSON files, reads the selected session file, and transmits it to the configured server endpoint via HTTP POST request. The server URL is configurable through VSCode settings, defaulting to localhost for development. Upon successful upload, the user receives confirmation including the count of stored events. The upload implementation is shown in Appendix E.5.

5.5 Lessons Learned

The integration process revealed several insights relevant for future GLSP tool integrations:

Action Protocol Variability. Different GLSP actions use inconsistent property names and structures. The tracking library needed multiple code paths to handle variations in how element positions, types, and identifiers are encoded across different action kinds.

For example, the six main action kinds use different naming conventions for the element or elements involved, as summarized in Table 5.1.

Table 5.1: Identifier fields used by different action kinds.

Action kind	Identifier field(s)
Create	`elementTypeId`
Edge	`sourceElementId`, `targetElementId`
Delete	`elementIds`
Update	`elementId`
Select	`selectedElementsIDs`, `deselectedElementsIDs`
Label edit	`labelId`

Viewport State Access. Capturing the current viewport state (scroll position, zoom level) required registering handlers at the GLSP client level, as this information is not directly accessible from the VSCode extension layer.

Compound Action Handling. GLSP's compound actions, which bundle multiple operations into a single transaction, required recursive processing to ensure all constituent actions were tracked.

Container ID Management. The tracking library needed special handling for container IDs in creation actions. The original implementation recorded the diagram root's UUID, which differed between recording and replay sessions. The solution was to use

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

a placeholder value (`$ROOT`) that signals the replay system to omit the container ID, letting GLSP use default behavior.

5.6 Challenges and Limitations

5.6.1 Performance Considerations

Profiling during development revealed that event serialization during high-frequency operations (like continuous element dragging) could introduce noticeable overhead. The implemented mitigations include debouncing for viewport and bounds tracking, asynchronous file operations for auto-save functionality, and lazy evaluation of event metadata extraction.

5.6.2 Privacy Considerations

The framework captures detailed interaction data that could potentially reveal sensitive information about modeling projects or user behavior. The implementation addresses this through several measures:

- **Opt-in Design.** Tracking must be explicitly started by the user.
- **Local Storage Default.** Session data is stored locally by default; upload to a remote server is a separate, explicit action.
- **Session Deletion.** The backend API supports session deletion for GDPR compliance.
- **Documentation.** Users can review exactly what data is being captured through the exported JSON files.

5.6.3 Type System Challenges

GLSP's element type identifiers follow tool-specific conventions. bigUML uses patterns like `CLASS__Class` and `ACTIVITY__OpaqueAction`, while other GLSP tools may use different formats. The tracking library implements flexible type matching to accommodate this variability, but this approach may require adjustment for other tools.

5.7 Summary

The integration of the tracking framework into bigUML demonstrates that comprehensive interaction tracking can be achieved in GLSP-based modeling tools with moderate development effort. The key success factors were leveraging GLSP's action protocol for semantic event capture, implementing proper lifecycle management for tracking sessions, and designing flexible schemas that accommodate the variability inherent in GLSP-based tools.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The resulting implementation provides complete coverage of diagram manipulation operations, property modifications, and viewport changes. Combined with the standalone eye-tracking application and the backend storage infrastructure, the framework establishes a foundation for empirical studies of modeling behavior and future development of analytical tools.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

User Experiments and Data Collection

To evaluate the practical applicability of the proposed interaction tracking framework, a small exploratory user study was conducted. The primary goal of this study was not to produce statistically significant results, but rather to demonstrate the framework's capability to capture, reconstruct, and analyze user interactions in a realistic modeling scenario. Seven participants completed the modeling task; six of those sessions were recorded with synchronized eye-tracking enabled.

6.1 Experimental Setup

Seven participants were recruited for this study. All participants had a technical background and basic familiarity with diagram-based modeling, though their experience levels varied. Each participant was asked to complete the same modeling task using the instrumented BIGUML editor.

The task consisted of recreating a predefined UML class diagram representing a simple university scenario. The diagram included four classes (*Person*, *Teacher*, *Student*, and *Course*), an inheritance hierarchy in which *Teacher* and *Student* both specialize *Person*, and two labeled associations connecting *Teacher* and *Student* to the courses they teach and attend, respectively. Each class defined a small set of attributes, so that participants would exercise the most common interaction patterns of a class-diagram editor – creating classes, adding attributes, drawing generalizations, and drawing associations with end labels. More advanced features such as interfaces, abstract classes, association classes, and aggregation or composition relationships were intentionally omitted to keep the task scoped to a typical introductory class diagram and the focus on interaction behavior rather than modeling complexity.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

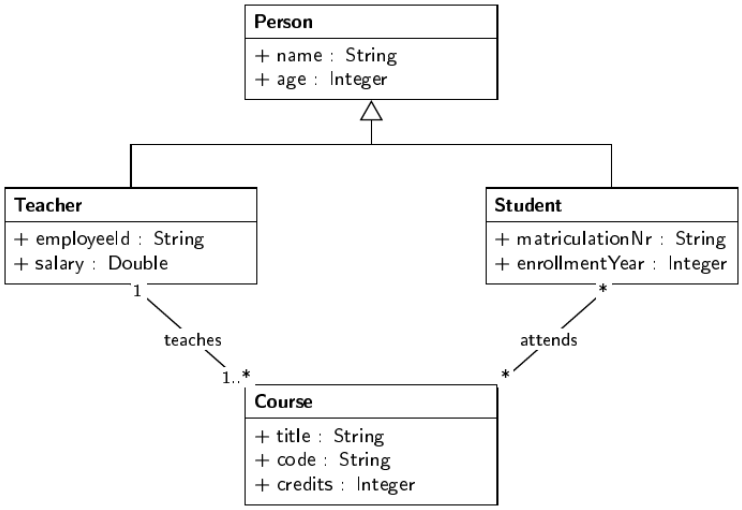

Figure 6.1: Reference UML class diagram used in the user study, representing a simple university scenario with an inheritance hierarchy and two labeled associations.

Before the task, participants received a short introduction to the basic operation of the tool, such as creating and editing model elements. The reference diagram itself was not demonstrated, and no recommended modeling sequence or strategy was provided. This was intended to minimize potential bias towards their modeling approach. Participants were then instructed to recreate the diagram as accurately as possible while interacting naturally with the tool. No time limit was imposed, and participants were not given step-by-step instructions, allowing for individual modeling strategies.

During each session, the tracking framework recorded:

- Low-level interaction events (e.g., clicks, drag-and-drop),
- Semantic modeling actions (e.g., element creation, connection creation, label editing),
- Viewport interactions (e.g., panning and zooming),
- Optional eye-tracking data (when enabled).

6.2 Data Collection Process

Each modeling session was recorded as a self-contained dataset consisting of interaction events and associated metadata. The frontend tracking library captured events in real time and exported them locally upon session completion. These session files were later uploaded to the backend system, where they were stored and made available for analysis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In addition to the recorded datasets, a demonstration video was created to illustrate the complete workflow of the system, including session recording, data export, backend ingestion, and analysis features. The video provides a practical overview of how the framework operates in an end-to-end scenario and serves as a supplementary artifact to this thesis[1].

The video showcases not only the recording process but also the replay functionality and analytical outputs, including clickstream reconstruction and heatmap visualization.

6.2.1 Dataset Availability

To support reproducibility and to enable other researchers to apply alternative analysis methods to the collected data, the dataset is published as an open archive on Zenodo[2].

The archive contains the unprocessed JSON exports produced by the frontend tracking library for each of the seven sessions, including the full interaction event stream, viewport changes, and – for the six eye-tracking-enabled sessions – the corresponding gaze point streams from the standalone eye-tracking application. No post-processing, filtering, or aggregation has been applied to these files; they correspond exactly to the data uploaded to the backend during the study. This deliberate design choice allows third parties to re-run the analyses presented below, to apply different analytical pipelines (e.g., process discovery in PM4Py or alternative friction heuristics), or to investigate aspects of the data not covered in this thesis. The persistent DOI resolves to the latest versioned release.

6.3 Analysis Approach

The collected data was analyzed on two levels: aggregated across all participants and individually per session.

Aggregated Analysis. Across all seven sessions, interaction data was combined to identify general behavioral patterns. This included:

- Frequency distributions of semantic actions (e.g., element creation, connections),
- Common action sequences derived from clickstream analysis,
- Aggregate heatmaps indicating frequently attended regions of the interface,
- Overall friction scores to identify recurring usability issues.

This aggregated view provides insight into shared modeling strategies and common interaction patterns across users.

[1] https://www.youtube.com/watch?v=DRvEqosXkcY, last accessed 2026-05-11
[2] https://doi.org/10.5281/zenodo.20112314, last accessed 2026-05-11

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Individual Session Analysis. In addition to aggregated metrics, each session was analyzed individually to capture user-specific behavior. The replay functionality enabled step-by-step reconstruction of each modeling process, allowing for a detailed inspection of interaction sequences. Specifically, this individual analysis focused on identifying distinct modeling strategies and the order of operations, noting the occurrence of friction indicators such as undo/redo bursts or navigation thrashing, measuring the time spent on specific tasks or elements, and evaluating differences in overall interaction efficiency between participants. Ultimately, this dual-level analysis demonstrates the flexibility of the framework in supporting both high-level pattern discovery and detailed qualitative inspection of user behavior.

6.4 Findings

The following sections summarize the analytical outputs obtained from the seven completed sessions. The intention is not to draw generalizable conclusions about BIGUML's usability – the sample is too small for that – but rather to demonstrate that the captured data can be transformed into meaningful, comparable, and actionable behavioral signals through the analytical pipeline described in Chapter 4. Throughout this section participants are referred to as P1 through P7, ordered chronologically by the start time of their session.

6.4.1 Session Overview

Table 6.1 summarizes the headline metrics for the seven sessions. All participants completed the modeling task and produced a diagram that closely matched the reference. Aggregated across the dataset, the backend has stored 1 440 interaction events spanning a total of approximately 27.4 minutes of active modeling time, with a per-session mean of 206 events and a mean duration of 3 min 55 s.

Table 6.1: Per-session summary of duration, captured events, dominant modeling strategy, composite friction score (0–100), the number of detected rage-click incidents (RC ev.), and the total number of rapid clicks they contained (RC clicks). No undo/redo bursts and no rapid-deletion bursts were detected in any session; navigation thrashing was detected only in P3 (1 incident).

ID	Dur. (s)	Events	ET	Strategy	Friction	RC ev.	RC clicks
P1	276.7	185	no	layout-centric	35.0	13	72
P2	351.4	155	yes	property-centric	35.0	8	35
P3	247.3	253	yes	layout-centric	39.5	19	78
P4	200.8	230	yes	layout-centric	35.0	21	99
P5	136.4	145	yes	property-centric	35.0	12	72
P6	141.3	165	yes	property-centric	35.0	13	70
P7	291.4	307	yes	layout-centric	35.0	28	124
Mean	235.0	205.7	—	—	35.6	16.3	78.6

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Two structural observations follow directly from the table. First, despite an identical task, both the number of recorded events (145 to 307) and the session duration (136.4 s to 351.4 s) varied by more than a factor of two between participants – a clear sign that even a tightly scoped modeling task elicits substantially different interaction profiles. Second, all seven sessions terminated cleanly with a `session_end` event and produced complete event streams, including the six gaze streams that were subsequently re-attached to their interaction sessions through the session identifier. This confirms that the tracking lifecycle and the eye-tracking integration described in Chapter 4 behave robustly under realistic conditions.

6.4.2 Sequential Modeling Patterns

Applying the n-gram analysis described in Section 2.4.1 to each session at the granularity `event_type:event_kind:element_type` reveals two qualitatively distinct modeling strategies that consistently appear across the seven participants. Throughout this subsection, classification is based on the relative weight of property-related events (`createNode:Property`, `applyLabelEdit`, `updateElementProperty`) versus layout-related events (`changeBounds`, `changeRoutingPoints`, `element_edit:compound`) within the filtered event stream of each session. It is important to note however that because of the low sample size these patterns are highly sensitive to individual user quirks.

Property-centric strategy (P2, P5, P6). For these three participants, the dominant local transitions are `createNode:Property` → `applyLabelEdit` (or `updateElementProperty`, depending on whether the participant edited the label inline or via the property palette) and its reverse.
P2 produced 13 consecutive `updateElementProperty` events, indicating long uninterrupted runs of attribute editing through the property panel. P5 represents the strongest version of this pattern: in 145 events, the session contains *not a single* layout-related event. Every class was placed in its default location and never moved; only properties and connections were added. P6 follows the same template, with one minor exception: a single `createEdge:Aggregation` was created and immediately deleted again, the only `deleteElement` event in the entire seven-session dataset. The shared workflow of these three participants can be summarized as: create class, immediately add and label all of its attributes, then move on to the next class. Connections are drawn only after most classes have been populated.

Layout-centric strategy (P1, P3, P4, P7). The remaining four participants display a completely different signature. The most frequent bigram for P1, P3, and P4 is `element_move:changeBounds` → `element_edit:compound` (12, 12, and 6 occurrences respectively), and the most frequent trigrams are typically the alternation `changeBounds` → `compound` → `changeBounds`. These participants alternate repeatedly between repositioning a class and committing the change, suggesting that

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

they prefer to fix the spatial layout of the diagram before completing the semantic content. P4 and P7 add an additional layer to this strategy through heavy use of `changeRoutingPoints` (6 and 9 occurrences respectively), indicating that they manually adjusted the path of generalization and association edges – something the property-centric participants did not do at all. P7 produced the layout-richest session in the dataset (11 `changeBounds`, 9 `changeRoutingPoints`, 9 `createNode:Property`), with the trigram `changeBounds` → `changeRoutingPoints` → `compound` occurring three times in succession.

Strategy and session duration. The strategy classification correlates loosely with session duration: the four layout-centric participants completed the task in a mean of 254 s, while the three property-centric participants completed it in a mean of 210 s. A 21 % slowdown for the layout-driven workflow is plausible given the additional repositioning and routing-point work, but the variance within each group (e.g., P5 at 136 s vs. P2 at 351 s, both property-centric) is large enough that any inference beyond "layout work takes additional time" is not warranted by the present sample.

Methodological observations. Even with seven participants, the sequence analysis already exposes that an apparently uniform task produces structurally different event streams, that the resulting strategies are observable as distinct n-gram signatures rather than as differences in raw event totals, and that within each strategy the participants converge on a small number of dominant transitions. This is encouraging from a tooling perspective: the chosen event granularity is fine enough to differentiate property-driven from layout-driven workflows, but coarse enough to keep the dominant n-grams stable and human-readable across participants. Methodologically, the result is consistent with the observation by van der Aalst [Aal16] that actual user behavior tends to deviate from the workflow assumed by tool designers, and that recording the real event sequence is a prerequisite for understanding modeling practice.

6.4.3 Friction Detection Results

All seven sessions produced a composite friction score in the lower band of the scale (six sessions at 35.0, P3 at 39.5; mean 35.6; interpretation category: *low friction*). The substructure of these scores, however, is the more informative finding.

Saturated rage-click component. For all seven participants, the rage-click sub-score reached its ceiling value of 100, while the undo/redo and rapid-deletion sub-scores remained at zero. The composite scores are therefore driven almost exclusively by rage-click events. The seven sessions accumulated 550 rapid clicks in total, distributed across 114 detected incidents (means: 78.6 clicks per session, 16.3 incidents per session). Inspection of the affected elements shows that the clicks consistently target node bodies of recently created `CLASS__Class` or `CLASS__Property` elements; the timing pattern (typically 4–6 clicks within 200–1500 ms) is most plausibly explained by participants

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

attempting to rename or re-select a freshly created element while the corresponding inline edit affordance was not yet active or not where they expected it.

This consistency across all seven sessions – regardless of whether the participant followed a property- or layout-centric strategy – strongly suggests that the threshold values calibrated for general-purpose web applications (3 clicks within 1500 ms, 50 px radius) are too lenient for diagram editors, where double-clicking and rapid multi-selection are part of normal interaction. The framework's parameterization (Section 2.4.4) deliberately keeps these thresholds configurable, and the data suggest that a higher minimum click count or a tool-specific exclusion of in-place edit affordances would be a sensible adjustment.

Severity escalation in P7. P7 is the only session in which rage-click clusters reached *critical* severity. Two clusters stand out: an 8-click sequence on a `CLASS__Class` element within 523 ms, and a 10-click sequence on a `CLASS__Property` element within 1 163 ms (the highest single-cluster click count in the dataset). Both clusters target elements in the upper part of the canvas and occur within a few seconds of each other, suggesting the participant was trying repeatedly to interact with an unresponsive or some not immediately apparent change on those elements. While anecdotal in isolation, this is exactly the kind of localized, time-bounded usability incident that the framework is intended to surface for follow-up investigation.

Spatial concentration of friction. The detector also reports the spatial centroids of all rage-click incidents. Aggregating the top hotspots across the seven sessions, four distinct clusters emerge that align with the layout of the reference diagram: a band around ($x \approx 1300$–$1400, y \approx 250$–300) corresponding to the *Person*/*Teacher*/*Student* row at the top, a left cluster around ($x \approx 900$–$1100, y \approx 400$–500) corresponding to the *Teacher* class, a right cluster around ($x \approx 1500$–$1700, y \approx 450$–550) corresponding to the *Student* class, and a bottom cluster around ($x \approx 1300$–$1400, y \approx 700$–750) corresponding to the *Course* class. Friction is therefore not distributed uniformly across the canvas but concentrates on the model elements that the participants were actively creating and editing, which matches the substantive interpretation given above and is consistent with the eye-tracking observations in Section 6.4.4.

Other indicators. No undo/redo bursts and no rapid-deletion bursts were detected in any of the seven sessions. P3 produced a single navigation-thrashing incident (six viewport changes within 2.2 s, total travel of ≈ 1452 pixels) shortly before drawing the last generalization edge; this matches the layout-centric strategy noted in the previous section, where viewport repositioning is more common than for the property-centric participants. The absence of undo/redo activity across all seven sessions is consistent with the relatively low task complexity – participants did not need to roll back substantial work – but it also means that this particular friction indicator could not be exercised in the present study.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.4.4 Visual Attention Patterns (Eye Tracking)

For the six sessions in which eye tracking was enabled (P2 through P7), the standalone WebGazer-based application produced gaze streams that were rendered as semi-transparent attention heatmaps overlaid on the editor screenshot, using the rendering pipeline described in Section 2.4.3. Figures 6.2, 6.3, and 6.4 show three of these overlays, chosen to span the two modeling strategies and the full range of session durations.

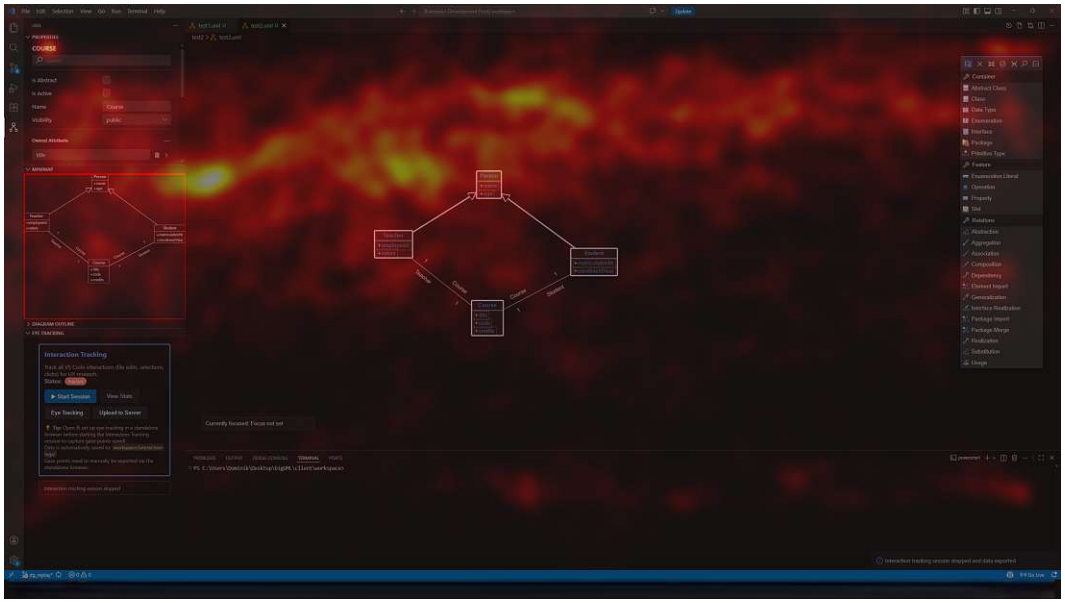

Figure 6.2: Aggregate gaze heatmap for participant P2 (property-centric, 351 s) overlaid on a screenshot of the BIGUML editor. Warmer colors indicate higher gaze density.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

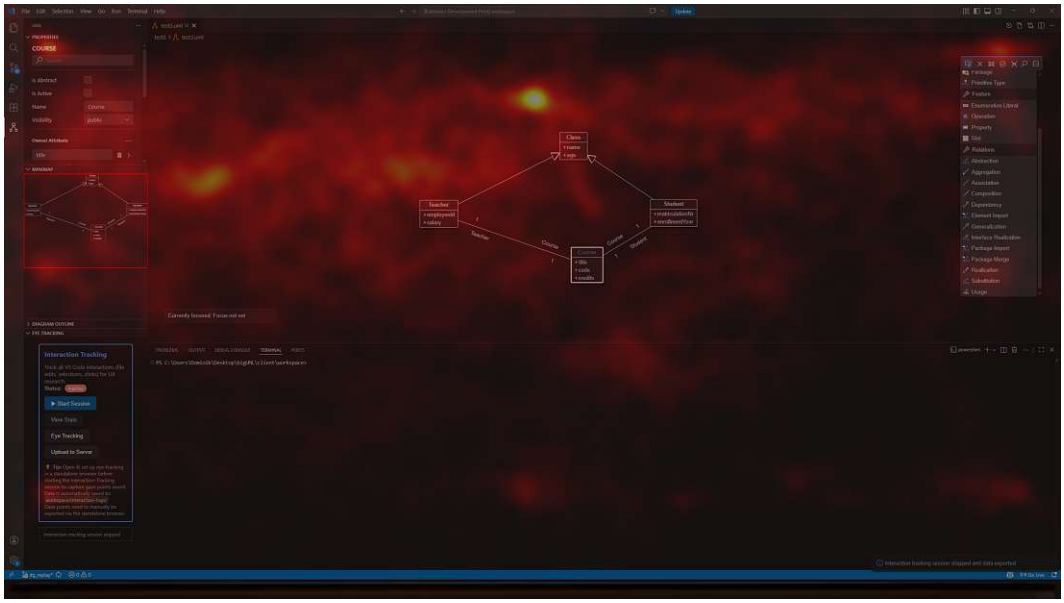

Figure 6.3: Aggregate gaze heatmap for participant P3 (layout-centric, 247 s) overlaid on a screenshot of the BIGUML editor.

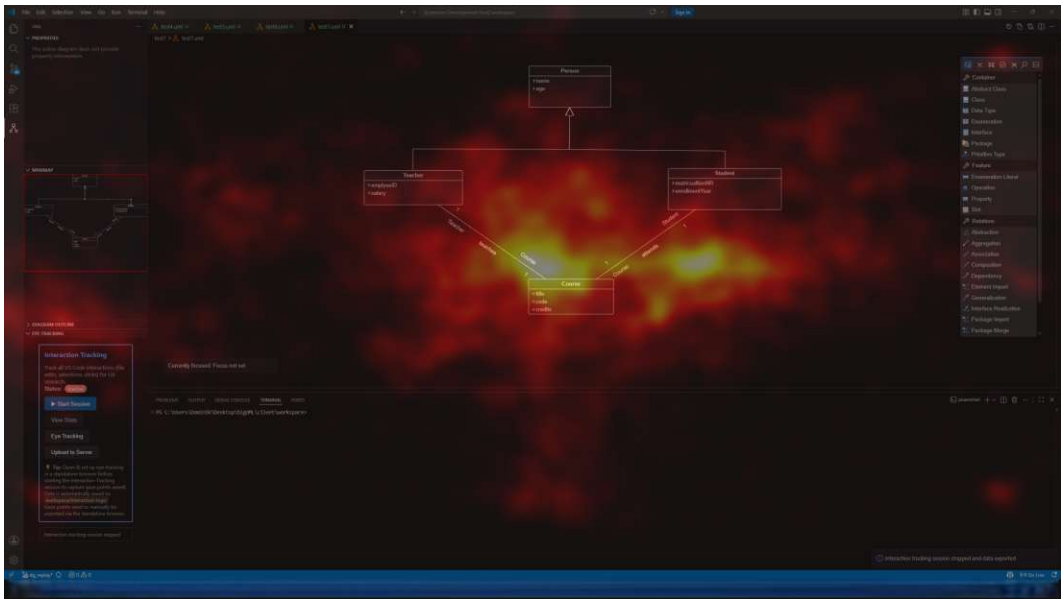

Figure 6.4: Aggregate gaze heatmap for participant P7 (layout-centric, 291 s, longest event-richest session) overlaid on a screenshot of the BIGUML editor. The strongest concentrations follow the inheritance hierarchy and the path of the *Course* associations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Three patterns are visible across the heatmaps. First, attention is heavily concentrated on the central canvas region where the diagram is being constructed; the highest density follows either the row of class headers (P2, P3) or the path of the labelled associations (P7). Second, the property panel on the left and the palette on the right receive only short bursts of attention rather than sustained fixation, indicating that these UI regions were used in a *glance-and-return* manner: participants briefly looked over to pick a tool or check a value and then returned to the canvas. Third, the layout-centric participants (P3 and especially P7) spread their attention across the inter-class regions – the labels, the routing points of the generalizations, the association names – substantially more than the property-centric participant (P2), whose hot zone is concentrated around the class bodies themselves. This is consistent with the sequence-analysis result that layout-centric participants spend additional time adjusting bounds and routing points, and demonstrates that the strategy distinction visible at the event-stream level also has a counterpart at the visual-attention level.

The latter observation is particularly relevant because it suggests that for this class of task the expensive screen real estate of the side panels is not where most of the cognitive load is borne; this is a hypothesis that a follow-up study with a larger and more controlled sample could confirm or refute.

It is worth restating the limitations highlighted in Section 2.4.3: the WebGazer accuracy of approximately 100–150 px is sufficient to identify region-level attention but not individual UI controls, the Gaussian smoothing introduces apparent continuity in regions where no fixations actually occurred, and any temporal dynamics are lost in the aggregate view. The heatmaps should therefore be read as overviews of *where* attention was distributed during the session, not as exact estimators of *what* was being looked at at any given moment.

6.4.5 Process-Mining Interoperability

In addition to the analysis methods implemented directly in the framework, the recorded interaction data can be exported in XES format and processed using external process-mining tools. To demonstrate this interoperability, XES logs generated from the study sessions were imported into ProM and Disco. Figure 6.5 shows a comparison of the event logs of participants P2 and P3 in ProM, while Figure 6.7 shows a process visualization generated from participant P7's session in Disco.

The purpose of these examples is not to perform an additional process-mining analysis of the study data, but to demonstrate that the event logs produced by the framework can be consumed by established process-mining software without additional transformation. The semantic modeling actions exported by the framework are interpreted as activities, while individual recording sessions correspond to process traces. Consequently, researchers can use the exported data as input for further analysis. This extends the analytical possibilities of the framework beyond the methods implemented in the backend itself.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

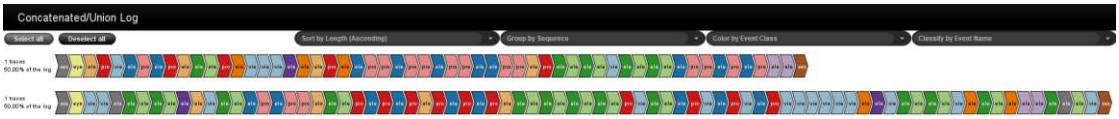

Figure 6.5: Participant sessions P2 and P3 imported into ProM from the framework's XES export. Each session is represented as a trace containing the chronologically ordered semantic modeling events.

Figure 6.5 shows the exported sessions of participants P2 and P3 after import into ProM. Each recording session is represented as an XES trace, while the semantic modeling interactions recorded by the framework appear as ordered events within that trace. Attributes exported alongside the events, such as timestamps and modeling-specific metadata, are available to ProM for subsequent process-mining analyses. The figure therefore demonstrates not only that the generated file can be imported, but also that the session and event structure of the recorded interaction data is preserved. The difference in trace length is also visually apparent, with P3 containing substantially more recorded events than P2.

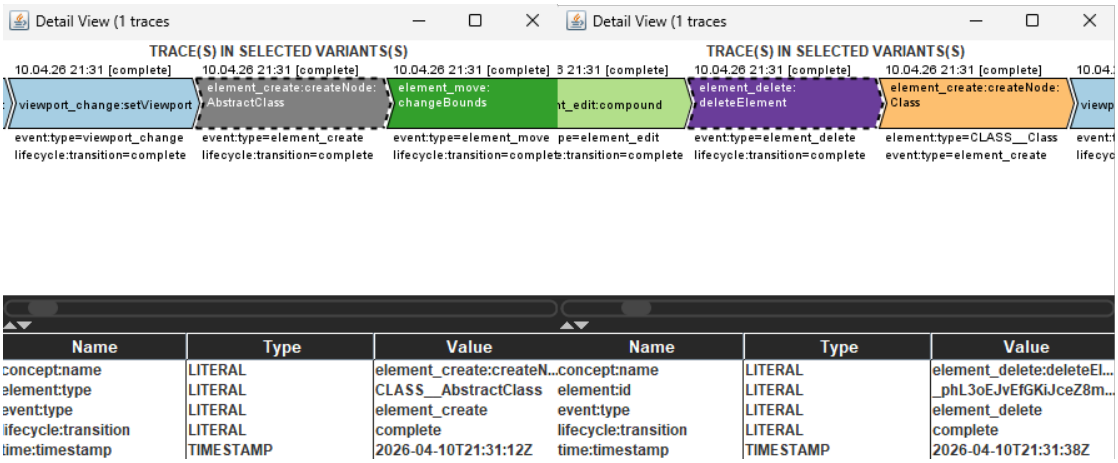

Figure 6.6: Detailed excerpt from participant P2's trace in ProM. The sequence shows the creation of an abstract class, which was not part of the reference model, and its subsequent deletion.

Figure 6.6 provides a closer view of individual events within participant P2's trace. The excerpt contains the creation of an abstract class that was not required by the reference model, followed later by its deletion. This illustrates that the semantic event names and their chronological ordering are retained in ProM and can be inspected at the level of individual modeling actions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

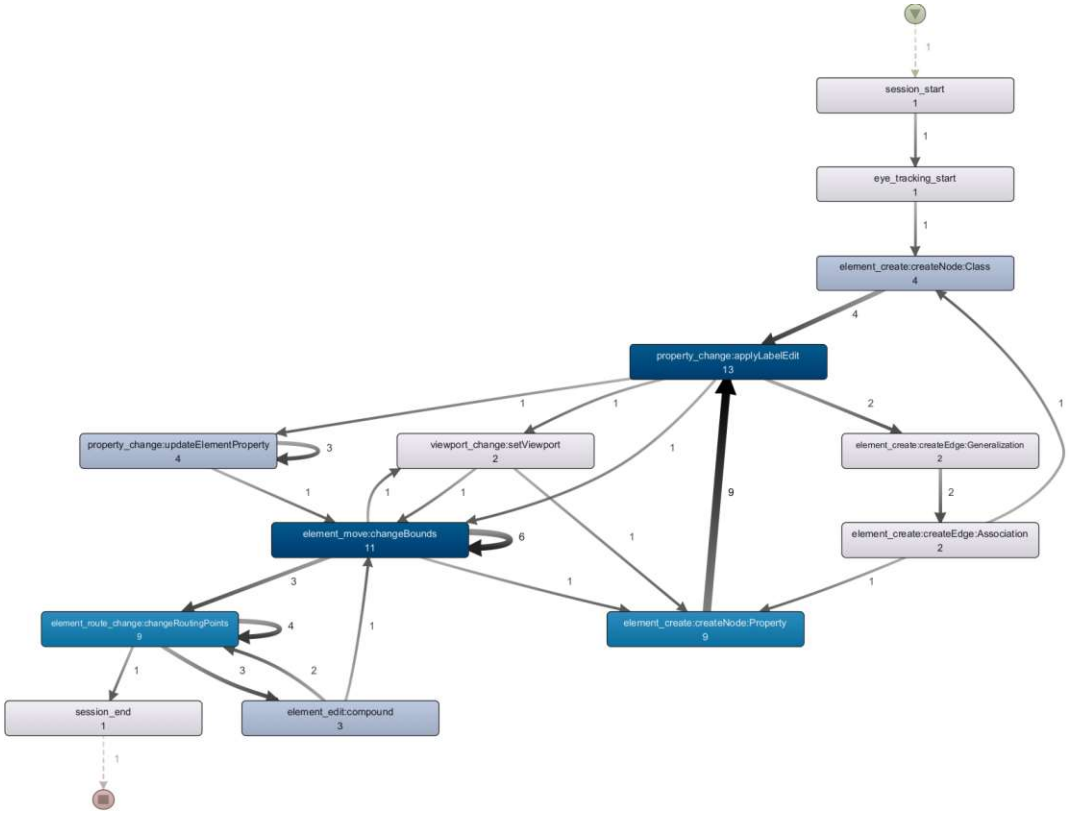

Figure 6.7: Process model generated by Disco from participant P7's exported XES session. Nodes represent recorded semantic modeling activities and edges represent observed transitions between consecutive activities.

Figure 6.7 shows the same type of exported interaction data visualized as a process model in Disco for participant P7. Activities in the graph correspond to semantic interaction events recorded by the tracking framework, while directed edges represent observed transitions between consecutive events. More prominent nodes and thicker connections correspond to activities and transitions that occurred more frequently in the recorded session. For example, the transition from *createNode:Property* to *property_change:applyLabelEdit* indicates that, in this session, newly created properties were consistently renamed immediately after creation. Although no process-mining interpretation is performed as part of this study, the visualization illustrates how the exported event stream can directly serve as input for process-discovery tools.

Together, these examples demonstrate interoperability at both levels: the exported sessions retain their event-level structure when inspected in ProM and can be used directly as input for process discovery in Disco.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

6.4.6 Synthesis

The seven sessions illustrate the kinds of insights the framework is intended to surface. From the sequence analysis, two qualitatively different modeling strategies (property-centric, observed in three sessions, versus layout-centric, observed in four) emerged consistently from a task that was deliberately simple and uniform; the strategies are visible as distinct n-gram signatures and correlate loosely with session duration. From the friction analysis, the dominant signal across the entire dataset is a saturated rage-click subscore that, on inspection of the underlying element data, points to an inline-edit affordance issue rather than to genuine user frustration with the tool as a whole – an actionable, concrete finding that a single session would not have been able to substantiate, and that P7 sharpens further by producing the dataset's only critical-severity clusters. From the eye-tracking analysis, the heatmaps confirm that attention is overwhelmingly canvas-centric and only briefly visits the side panels, and they show that the strategy distinction observed at the event-stream level has a clear counterpart at the visual-attention level. None of these observations would have been visible from session timing alone, which is the central claim of the framework and which the present dataset substantiates at the proof-of-concept level.

6.5 Discussion of Study Scope

It is important to emphasize that the analyses presented in Section 6.4 are based on a small-scale exploratory evaluation. Seven sessions are sufficient to demonstrate that the framework captures, structures, and analyzes interaction data as intended, and they are sufficient to surface concrete tool-design hypotheses (the friction-threshold recalibration, the strategy-versus-attention correspondence, the side-panel under-utilization). They are not, however, sufficient to support quantitative claims about the prevalence of either modeling strategy in the broader user population, nor about the absolute level of friction in BIGUML relative to other modeling tools.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Within those caveats, the study serves its intended purpose as a proof of concept for the framework, demonstrating that:

- the framework can reliably capture complete interaction sessions, including their full event stream, viewport history, and – where enabled – a synchronized gaze stream;
- the recorded data can be transformed into meaningful analytical outputs such as comparative session statistics, n-gram frequency tables, friction indicators with spatial localization and severity grading, and overlay heatmaps;
- both the aggregated and the individual perspectives surface non-trivial behavioral information – for example the property- versus layout-centric strategies, the saturated rage-click component, the rage-click hotspot clusters, the strategy-attention correspondence visible in the heatmaps – that would be difficult to recover from raw session timing or post-hoc questionnaires alone;
- the published Zenodo dataset (Section 6.2.1) makes both the raw data and the analytical pipeline reproducible by third parties, which is itself a contribution that the small sample size does not undermine.

Future work could extend this evaluation with a larger participant pool, more complex modeling tasks, and controlled experimental conditions to enable quantitative validation of usability improvements. Three directions are particularly motivated by the findings reported above. First, the saturated rage-click signal across all seven participants suggests that a recalibration study of the friction thresholds for diagram-editor workloads is a prerequisite for using the absolute friction scores in cross-tool comparisons. Second, the consistent property- versus layout-centric split, combined with its visible counterpart in the eye-tracking data, suggests a focused study of how modeling strategy interacts with task complexity and modeler experience. Third, the canvas-centric attention distribution invites a targeted investigation of how often participants actually need to leave the canvas to consult the property and palette panels, and whether co-locating the most frequently used affordances with the canvas would reduce that travel.

6.6 Evaluation Against Requirements

In addition to examining the behavioral findings obtained from the user study, the implemented framework can be evaluated against the functional and non-functional requirements derived in Chapter 3. This assessment distinguishes between requirements that were directly exercised during the reference integration and user study, and requirements whose fulfillment is supported primarily by the architectural design or implementation. The resulting assessment is summarized in Table 6.2.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 6.2: Evaluation of the framework against the requirements defined in Chapter 3.

ID	Requirement	Evaluation Evidence	Assessment
FR1	Low-level interaction capture	Mouse and interaction events were captured throughout all seven completed sessions and persisted as complete event streams.	Fulfilled
FR2	Semantic action capture	GLSP actions for element creation, deletion, label editing, connection creation, property changes, and element movement were translated into semantic interaction events and used in the subsequent analyses.	Fulfilled
FR3	Viewport interaction capture	Pan, zoom, and viewport changes were captured and stored as part of the session history.	Fulfilled
FR4	Eye-tracking support	Six of the seven study sessions included gaze streams that could be synchronized with the corresponding interaction sessions and visualized as heatmaps.	Fulfilled
FR5	Standardized event representation	Recorded actions were transformed into the common interaction-event schema while the JSONB payload preserved tool-specific information.	Fulfilled
FR6	Storage and session import	Complete session files were uploaded through the REST API, persisted in PostgreSQL, and retrieved for analysis.	Fulfilled
FR7	Behavioral analysis	The study successfully applied n-gram analysis, friction detection, and gaze heatmaps to recorded sessions and produced interpretable behavioral findings.	Fulfilled
FR8	Interaction replay	Recorded modeling sessions could be reconstructed from their event streams using the replay component, which was also used to validate tracking completeness.	Fulfilled
FR9	External analysis support	Study sessions were exported as XES and successfully imported into ProM and Disco, demonstrating interoperability with external process-mining tools.	Fulfilled
NFR1	Modularity and extensibility	Tracking, eye tracking, storage, replay, and analysis are implemented as separate components, while the event schema permits additional event types and metadata.	Fulfilled
NFR2	Tool agnosticism	The backend uses generic REST contracts and a flexible event schema, and a third-party integration guide is provided in Appendix F. However, the implementation has only been empirically evaluated with bigUML.	Partially validated
NFR3	Minimal performance impact	Debouncing, batching, and asynchronous processing reduce overhead, and the framework remained usable during the study. However, no controlled quantitative performance benchmark was conducted.	Partially validated
NFR4	Privacy and security	Tracking is opt-in, session data is stored locally by default, remote upload is explicit, API access is protected, and sessions can be deleted. A complete data-protection impact assessment was outside the scope of the thesis.	Partially validated
NFR5	Web compatibility	The frontend is implemented using TypeScript and web-based GLSP APIs and was successfully operated inside the VS Code/webview-based bigUML environment.	Fulfilled
IR1	GLSP reference integration	The complete framework was integrated into the GLSP-based bigUML editor and used in the seven-participant evaluation.	Fulfilled

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Overall, the evaluation provides direct evidence for the fulfillment of the functional requirements. The complete interaction streams recorded during the study demonstrate that low-level interactions, semantic modeling operations, and viewport changes can be captured and merged. The six sessions with eye-tracking further demonstrate that gaze data can be associated with the same modeling sessions. The recorded data could also be replayed and processed by the implemented clickstream, friction, and heatmap analyses. Finally, the successful import of the generated XES logs into ProM and Disco shows that the collected data is not limited to the analysis functionality implemented by the framework itself.

The non-functional requirements are supported to different degrees. Modularity and extensibility are directly reflected in the separation between the frontend tracking library, the standalone eye-tracking component, the backend, and the analytical modules, as well as in the extensible event representation. Tool agnosticism, however, has so far been demonstrated primarily at the architectural level. The backend does not depend on bigUML-specific structures and the integration guide defines a generic contract for additional frontends, but bigUML remains the only modeling tool into which the framework has been fully integrated and evaluated. Consequently, this requirement should be considered supported by the design but not yet empirically validated across different modeling environments.

A similar distinction applies to performance and privacy. The implementation contains explicit mechanisms for reducing runtime overhead, such as debouncing and asynchronous processing, and no blocking performance issue prevented the framework from being used during the study. Nevertheless, the evaluation did not include a controlled measurement of latency or resource consumption with and without tracking enabled. Privacy requirements are addressed through explicit opt-in, local storage by default, transparent exports, authenticated backend access, and session deletion. These mechanisms provide evidence that the privacy-by-design goals were considered in the implementation, although a full data-protection impact assessment and production-level security evaluation remain outside the scope of this work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 7

Conclusion

This thesis investigated the design, implementation, and evaluation of a modular, open-source framework for tracking user interactions in web-based modeling tools. Motivated by the observation that web-based modeling environments lack purpose-built infrastructure for empirical user-experience research, and that existing behavioral-analytics platforms either cannot capture the semantic structure of diagrammatic editing or are tied to specific research lab tools, the work followed a Design Science Research methodology to produce four interconnected artefacts: a TypeScript frontend tracking library, a standalone WebGazer-based eye-tracking application, a Python/FastAPI backend with REST endpoints for ingestion and analysis, and a PostgreSQL schema optimized for timestamped event streams. The framework was integrated into the GLSP-based BIGUML editor as a reference implementation and exercised in a small exploratory user study with seven participants.

7.1 Summary of Contributions

The principal contributions of this work are technical, methodological, and empirical. On the technical side, the framework demonstrates that tool-agnostic capture of *semantic* modeling operations is feasible by hooking into the GLSP action protocol rather than relying on raw DOM events. Diagram manipulations, property modifications, and viewport changes are captured with full coverage in BIGUML, and the JSONB-backed event schema is generic enough to admit additional GLSP-based editors without changes to the backend. The interaction replay component, although it originated as a debugging aid, evolved into a self-contained capability for reconstructing modeling sessions from their event log and proved instrumental in validating tracking completeness.

Methodologically, the work provides a structured analytical pipeline that turns raw event streams into three complementary forms of behavioral information: configurable

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

n-gram clickstream analysis with XES export for downstream process mining, a four-component composite friction score (rage clicking, undo/redo bursts, navigation thrashing, rapid deletion), and aggregated gaze heatmaps with optional screenshot overlay. Each component is parameterized so that researchers can tune granularity, smoothing, and detection thresholds to the specific question at hand.

Empirically, the case study, even at its current scale, demonstrated that the framework surfaces non-trivial behavioral patterns. Two qualitatively different modeling strategies emerged from an identical task – a *property-centric* workflow and a *layout-centric* workflow – and the friction analysis flagged a saturated rage-click signal that, on inspection, pointed to a concrete inline-edit affordance issue rather than to general user frustration. The complete dataset has been deposited on Zenodo [Gra26], enabling third-party replication of the analyses presented in Chapter 6 and the application of alternative analytical pipelines.

7.2 Revisiting the Research Questions

RQ1 – Which user interactions are of importance in a graphical modeling tool and ought to be collected? The literature review and the framework design distinguish three layers that should be captured jointly: (i) low-level pointer and keyboard events as the substrate for general usability metrics such as rage clicks, (ii) semantic modeling actions (element creation, deletion, edge insertion, label edits, property changes) that reflect actual user intent, and (iii) view-manipulation events (pan, zoom, scroll) that document navigation strategy. The implemented `InteractionEventType` taxonomy enumerates all three layers, and the case-study results confirm that the semantic and view-manipulation layers are necessary: the property-centric versus layout-centric distinction would be invisible from raw click streams alone.

RQ2 – How can collected interaction data be structured, stored, and analyzed efficiently to provide actionable insights? The answer is twofold. Structurally, a hybrid schema that combines fixed columns for session metadata and frequently queried event attributes with a JSONB payload for the full event body offers both efficient indexed access and lossless preservation of tool-specific detail. Analytically, three pipelines were implemented and validated on real session data: configurable n-gram analysis with transition matrices and IEEE 1849-2016 XES export for compatibility with PM4Py, ProM, and Disco; sliding-window friction detection with weighted composite scoring; and on-demand Gaussian-smoothed heatmap generation with optional screenshot overlay. The case-study findings – the modeling-strategy taxonomy, the rage-click hotspot clusters, and the canvas-centric attention distribution – illustrate that these pipelines produce actionable output rather than only descriptive statistics.

RQ3 – How can eye-tracking be effectively integrated into existing web-based modeling tools using different tracking technologies? The architectural answer

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

that emerged during development is to keep the eye-tracking subsystem out of the host application's webview and to integrate at the data layer instead. The initial attempt to embed WebGazer directly into the BIGUML webview ran into Content-Security-Policy restrictions, real-time performance degradation, and a partial-screen calibration that was unsuitable for the actual tracking task. Refactoring eye-tracking into a standalone browser application with full-screen calibration, isolated processing, and a session-identifier-based join with the interaction event stream resolved all three problems and, as a side effect, made it straightforward to substitute WebGazer with another tracker in future iterations.

7.3 Limitations

Several limitations should be made explicit. First, the user study currently rests on seven sessions. The reported strategy distinction, friction interpretation, and gaze patterns must therefore be read as illustrative case-by-case findings rather than as established trends and would ultimately require a larger and controlled follow-up study.

Second, the friction-detection thresholds were inherited from values calibrated for general-purpose web applications, and the case study itself produced evidence that they are too lenient for diagram editors: all sessions saturated the rage-click sub-score, predominantly because of normal in-place edit interactions. The framework retains its utility because thresholds are configurable, but a recalibration study specific to diagrammatic editing is a prerequisite for using the absolute friction scores for cross-tool comparison.

Third, the eye-tracking integration is not yet end-to-end automated. Gaze data must currently be exported manually from the standalone application and uploaded to the backend, and the standalone application captures screen coordinates that have not yet been transformed into diagram-canvas coordinates accounting for viewport pan and zoom. WebGazer's accuracy of approximately 100–150 px is sufficient for region-level attention analysis but inadequate for resolving fixations to individual model elements in dense diagrams.

Fourth, the framework has been validated only against BIGUML. Although the tool-agnostic ambition is reflected in the schema design, the action interception, and the absence of BIGUML-specific assumptions in the backend, additional integrations – for instance, with another GLSP-based editor or with a non-GLSP web modeling tool – would be required to substantiate that ambition empirically.

Finally, the privacy story is sound at the design level (opt-in tracking, local-by-default storage, GDPR-compliant deletion endpoints, transparent JSON exports), but a full data-protection impact assessment was outside the scope of this work and would be necessary before any production deployment in a study with non-consenting bystanders or with sensitive modeling content.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.4 Future Work

Building on the present results, four directions appear particularly promising. The most immediate is the completion of more user studies, which will enable a more robust comparison of modeling strategies and a meaningful baseline for the friction-threshold recalibration. A focused recalibration experiment, in which the rage-click parameters are systematically swept and validated against ground-truth task-difficulty annotations, would directly address the saturation effect documented in Section 6.4.3.

A second direction is the closure of the eye-tracking pipeline: automatic upload of gaze streams alongside the corresponding interaction session, transformation of screen coordinates to canvas coordinates with full pan-and-zoom compensation, and the addition of fixation detection at ingestion time would move the analysis from raw gaze density to fixation-based attention metrics that are directly comparable with the eye-tracking literature surveyed in Section 2.4.2.

A third direction is the integration of higher-order analytical methods. The XES export already opens the door to process discovery via PM4Py and ProM; adding conformance checking against a reference modeling workflow, performance analysis on time-to-completion per element, and – in a longer perspective – predictive modeling of task abandonment from early-session features would each turn the framework from an observatory into a diagnostic instrument. The kinematic features pioneered by Interaction Lab [FLDAGRT22] are an obvious complement on the input side.

A fourth direction is broader ecosystem validation. Integrating the tracking library into a second GLSP-based editor and into a non-GLSP web modeling tool would validate the tool-agnostic claim, and a comparative evaluation against the closest related systems – TOOL for the lab-study scenario and MER for the EMF-based change-event scenario – would situate the framework within the existing landscape of modeling-aware behavioral-analytics tools. To ease any further work into this direction, a detailed integration-guide is provided in Appendix F.

7.5 Closing Remarks

User-experience research on graphical modeling tools has historically been hampered by the absence of a tracking infrastructure that captures the right events at the right level of abstraction and that can be deployed without rewriting the host editor. The framework presented in this thesis is an attempt to fill that gap: a tool-agnostic, open-source library and backend that records interactions both as low-level events and as semantic modeling actions, complements them with optional gaze data, and turns the resulting streams into clickstream, friction, and attention analyzes that are directly actionable for tool designers. Even the small case study reported here surfaced concrete, design-relevant findings that would have been invisible from session timing alone, which is the central proposition of the work and the basis on which subsequent, larger-scale studies of modeling behaviour can build.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Overview of Generative AI Tools Used

During the development of this thesis, the following AI-assisted tools were used:

- **Claude (Anthropic):** The AI assistant Claude was used for generating initial boilerplate code for the FastAPI backend infrastructure, including basic endpoint structures, Pydantic schema definitions, and SQLAlchemy model templates. Additionally, Claude was consulted during debugging sessions to analyze console log outputs, identify potential error sources, and suggest solutions for runtime issues encountered during development.
- **Overleaf AI Assistant:** The integrated AI assistant in Overleaf was used to rephrase and improve the clarity of individual paragraphs throughout the written thesis.

All AI-generated content was critically evaluated and manually verified and adapted before incorporation into the final work. The conceptual design, architectural decisions, implementation logic and argumentations remain the work of the author.

Additional Tools: Reference management and citation formatting were handled using Zotero with Overleaf integration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX A

Database Schema DDL

This appendix contains the SQL DDL for the database schema referenced in Section 4.1.2.

A.1 Recording Sessions Table

Listing A.1: Recording Sessions table structure

```
1  CREATE TABLE recording_sessions (
2      id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
3      session_id VARCHAR(255) UNIQUE NOT NULL,
4
5      -- Tool identification
6      tool_id VARCHAR(100) NOT NULL DEFAULT 'unknown',
7      tool_version VARCHAR(50),
8
9      -- Editor/Model information
10     editor_type VARCHAR(100),
11     model_file VARCHAR(255),
12     model_file_path TEXT,
13
14     -- User information
15     user_name VARCHAR(255),
16     workspace VARCHAR(255),
17
18     -- Timing
19     start_time TIMESTAMP WITH TIME ZONE NOT NULL,
20     end_time TIMESTAMP WITH TIME ZONE,
21     duration_ms INTEGER,
22     total_events INTEGER DEFAULT 0,
23
24     -- Metadata
25     created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    extra_data JSONB
);
```

A.2 Interaction Events Table

Listing A.2: Interaction Events table structure

```
CREATE TABLE interaction_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES recording_sessions(id)
        ON DELETE CASCADE,

    -- Timing
    timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
    sequence_number INTEGER NOT NULL,

    -- Event classification
    event_type VARCHAR(50) NOT NULL,
    event_kind VARCHAR(100),

    -- Denormalized element information
    element_id VARCHAR(255),
    element_type VARCHAR(255),
    position_x FLOAT,
    position_y FLOAT,

    -- Full event data
    data JSONB NOT NULL
);
```

A.3 Gaze Points Table

Listing A.3: Gaze Points table structure

```
CREATE TABLE gaze_points (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES recording_sessions(id)
        ON DELETE CASCADE,

    -- Timing
    timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
    sequence_number INTEGER NOT NULL,

    -- Position (screen coordinates)
    x FLOAT NOT NULL,
    y FLOAT NOT NULL,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
13
14      -- Normalized coordinates (0-1)
15      x_normalized FLOAT,
16      y_normalized FLOAT,
17
18      -- Quality metrics
19      pupil_diameter FLOAT,
20      confidence FLOAT,
21
22      -- Fixation grouping (computed post-hoc)
23      fixation_id INTEGER,
24      is_fixation BOOLEAN,
25
26      -- Eye source
27      eye VARCHAR(10),
28
29      -- Raw tracker data
30      raw_data JSONB
31  );
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX B

Frontend Tracking Library Code

This appendix contains the implementation details for the frontend tracking library described in Section 4.3.

B.1 InteractionTracker Service Initialization

Listing B.1: InteractionTracker service initialization

```
@injectable()
export class InteractionTracker {
    private events: InteractionEvent[] = [];
    private currentSession?: SessionData;
    private isTracking = false;
    private disposables: vscode.Disposable[] = [];

    // Viewport change debouncing
    private lastViewportEvent: { timestamp: number; data: any } |
        null = null;
    private viewportDebounceTimeout: NodeJS.Timeout | null = null;
    private readonly VIEWPORT_DEBOUNCE_MS = 100;

    @inject(TYPES.ActionListener)
    protected readonly actionListener?: any;

    public startSession(sessionId?: string): void {
        // Initialize session with tool metadata
        const editorType = this.detectDiagramType();
        const modelFilePath = this.getActiveUmlFilePath();

        this.currentSession = {
            sessionId: sessionId || this.generateSessionId(editorType
                ),
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
            startTime: Date.now(),
            toolId: 'bigUML',
            toolVersion: '0.6.3',
            editorType: editorType,
            modelFile: modelFilePath ? path.basename(modelFilePath) :
                undefined,
            user: process.env.USER || 'unknown',
            workspace: vscode.workspace.workspaceFolders?.[0]?.name
        };

        this.setupTracking();
        this.isTracking = true;
    }
}
```

B.2 Interaction Event Type Enumeration

Listing B.2: Interaction event type enumeration

```
export enum InteractionEventType {
    // Diagram interactions
    ELEMENT_SELECT = 'element_select',
    ELEMENT_CREATE = 'element_create',
    ELEMENT_DELETE = 'element_delete',
    ELEMENT_MOVE = 'element_move',
    ELEMENT_RESIZE = 'element_resize',
    ELEMENT_EDIT = 'element_edit',

    // View changes
    VIEWPORT_CHANGE = 'viewport_change',
    ZOOM_CHANGE = 'zoom_change',
    SCROLL = 'scroll',

    // Property modifications
    PROPERTY_CHANGE = 'property_change',

    // File operations
    FILE_OPEN = 'file_open',
    FILE_SAVE = 'file_save',
    FILE_CLOSE = 'file_close',
    TEXT_EDIT = 'text_edit',

    // Session lifecycle
    SESSION_START = 'session_start',
    SESSION_END = 'session_end'
}
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

B.3 GLSP Action Listener Registration

Listing B.3: GLSP action listener registration

```
private setupTracking(): void {
    if (this.actionListener) {
        // Listen to CLIENT actions (from diagram webview)
        const glspClientDisposable = this.actionListener.
            registerListener(
            (message: any) => {
                this.processGLSPAction(message.action);
            }
        );
        this.disposables.push(glspClientDisposable);

        // Listen to SERVER responses
        const glspServerDisposable =
            this.actionListener.registerServerListener?.((message:
                any) => {
                this.processGLSPAction(message.action);
            });
        if (glspServerDisposable) {
            this.disposables.push(glspServerDisposable);
        }
    }
}
```

B.4 GLSP Action Processing

Listing B.4: GLSP action processing

```
private processGLSPAction(action: any): void {
    if (!action || !action.kind) return;

    if (action.kind === 'createNode') {
        this.trackEvent(InteractionEventType.ELEMENT_CREATE, {
            kind: action.kind,
            elementTypeId: action.elementTypeId,
            containerId: '$ROOT',
            location: action.location
        });
    }
    else if (action.kind === 'createEdge') {
        this.trackEvent(InteractionEventType.ELEMENT_CREATE, {
            kind: action.kind,
            elementTypeId: action.elementTypeId,
            sourceElementId: action.sourceElementId,
            targetElementId: action.targetElementId
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
        });
    }
    else if (action.kind === 'elementSelected') {
        this.trackEvent(InteractionEventType.ELEMENT_SELECT, {
            kind: action.kind,
            selectedElementsIDs: action.selectedElementsIDs,
            deselectedElementsIDs: action.deselectedElementsIDs
        });
    }
    else if (action.kind === 'applyLabelEdit') {
        this.trackEvent(InteractionEventType.PROPERTY_CHANGE, {
            kind: action.kind,
            labelId: action.labelId,
            text: action.text
        });
    }
    else if (action.kind === 'changeBounds') {
        this.trackEvent(InteractionEventType.ELEMENT_MOVE, {
            kind: 'changeBounds',
            newBounds: action.newBounds
        });
    }
    // Additional action handlers...
}
```

B.5 Debounced Viewport Tracking

Listing B.5: Debounced viewport tracking

```
private trackViewportEvent(data: any): void {
    if (!this.isTracking || !this.currentSession) return;

    // Clear any pending debounce timeout
    if (this.viewportDebounceTimeout) {
        clearTimeout(this.viewportDebounceTimeout);
    }

    // Store the latest viewport data
    this.lastViewportEvent = {
        timestamp: Date.now(),
        data
    };

    // Set debounce timeout - record after user stops panning/zooming
    this.viewportDebounceTimeout = setTimeout(() => {
        if (this.lastViewportEvent) {
            this.trackEvent(
                InteractionEventType.VIEWPORT_CHANGE,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
                this.lastViewportEvent.data
            );
            this.lastViewportEvent = null;
        }
        this.viewportDebounceTimeout = null;
    }, this.VIEWPORT_DEBOUNCE_MS);
}
```

B.6 ViewportTrackingHandler in GLSP Client

Listing B.6: ViewportTrackingHandler in GLSP client

```
@injectable()
export class ViewportTrackingHandler implements IActionHandler {
    @inject(TYPES.IActionDispatcher)
    protected actionDispatcher: IActionDispatcher;

    handle(action: Action): ICommand | Action | void {
        if (SetViewportAction.is(action)) {
            const viewport = action.newViewport;
            if (viewport) {
                this.trackViewportChange(viewport.scroll, viewport.
                    zoom);
            }
        }
        return undefined; // Don't consume - let it propagate
    }
}
```

B.7 ElementBoundsTrackingHandler

Listing B.7: ElementBoundsTrackingHandler

```
@injectable()
export class ElementBoundsTrackingHandler implements IActionHandler {
    private pendingBounds: Map<string, {
        position: { x: number; y: number };
        size: { width: number; height: number }
    }> = new Map();

    handle(action: Action): ICommand | Action | void {
        if (ChangeBoundsOperation.is(action)) {
            if (action.newBounds && action.newBounds.length > 0) {
                for (const bound of action.newBounds) {
                    if (bound.newPosition && bound.newSize) {
                        this.trackBoundsChange(
                            bound.elementId,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
                                bound.newPosition,
                                bound.newSize
                            );
                        }
                    }
                }
            }
            return undefined;
        }
}
```

B.8 Session Export Implementation

Listing B.8: Session export implementation

```
private exportSession(): void {
    if (!this.currentSession) return;

    // Convert timestamps to ISO format for export
    const sessionWithIsoTimestamps = {
        ...this.currentSession,
        startTime: new Date(this.currentSession.startTime).
            toISOString(),
        endTime: this.currentSession.endTime
            ? new Date(this.currentSession.endTime).toISOString()
            : undefined
    };

    const eventsWithIsoTimestamps = this.events.map(event => ({
        ...event,
        timestamp: new Date(event.timestamp).toISOString()
    }));

    // Export JSON
    const jsonFilePath = path.join(this.logDir, `${sessionId}.json`);
    fs.writeFile(jsonFilePath, JSON.stringify({
        session: sessionWithIsoTimestamps,
        events: eventsWithIsoTimestamps
    }, null, 2), 'utf-8', err => {
        if (err) console.error('Failed to export JSON:', err);
    });

    // Export CSV for analysis tools
    const csvFilePath = path.join(this.logDir, `${sessionId}.csv`);
    const csvContent = this.convertToCSV();
    fs.writeFile(csvFilePath, csvContent, 'utf-8', err => {
        if (err) console.error('Failed to export CSV:', err);
    });
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
}
```

B.9 Diagram Type Detection

Listing B.9: Diagram type detection

```
private detectDiagramType(): string {
    const activeTab = vscode.window.tabGroups.activeTabGroup?.
        activeTab;
    if (activeTab?.input) {
        const input = activeTab.input as any;
        if (input.uri) {
            const fileName = input.uri.fsPath || '';
            const match = fileName.match(
                /\.(activity|class|communication|deployment|
                    information_flow|package|state_machine|
                    use_case|sequence)\.uml$/i
            );
            if (match) {
                return match[1].toLowerCase();
            }
        }
    }
    return 'unknown';
}
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX C

Eye-Tracking Integration Code

This appendix contains the implementation details for the eye-tracking integration described in Section 4.4.

C.1 Standalone Eye-Tracking Application Structure

Listing C.1: Standalone eye-tracking application structure

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Eye Tracking Demo</title>
    <script src="webgazerv2.js"></script>
</head>
<body>
    <div class="container">
        <!-- Status display -->
        <div class="status-section">
            <span id="statusIndicator" class="status-inactive">
                Inactive</span>
            <div>Gaze Points Collected: <span id="gazeCount">0</span>
                </div>
        </div>

        <!-- Controls -->
        <button id="startBtn">Start Eye Tracking</button>
        <button id="calibrateBtn">Calibrate</button>
        <button id="exportBtn">Export Data</button>

        <!-- Calibration overlay -->
        <div id="calibrationSection" class="calibration-section">
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
            <div class="calibration-grid">
                <!-- 9-point calibration grid -->
            </div>
        </div>
    </div>
</body>
</html>
```

C.2 WebGazer Initialization

Listing C.2: WebGazer initialization

```
async startEyeTracking() {
    await webgazer
        .setRegression('ridge')
        .setTracker('TFFacemesh')
        .setGazeListener((data) => {
            if (data && data.x && data.y) {
                this.handleGazeData(data);
            }
        })
        .begin();

    webgazer.showVideo(true);
    webgazer.showPredictions(true);

    this.isActive = true;
}

handleGazeData(data) {
    const gazePoint = {
        x: data.x,
        y: data.y,
        timestamp: Date.now()
    };

    this.gazeData.push(gazePoint);

    // Visual feedback
    this.showGazePoint(data.x, data.y);
}
```

C.3 Calibration Point Handling

Listing C.3: Calibration point handling

```
calibratePoint(pointElement) {
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    const index = parseInt(pointElement.dataset.index);

    // Only allow clicking in sequence
    if (index !== this.currentCalibrationPoint) {
        this.updateStatus('Please click the highlighted dot');
        return;
    }

    const rect = pointElement.getBoundingClientRect();
    const x = rect.left + rect.width / 2;
    const y = rect.top + rect.height / 2;

    // Record multiple samples for accuracy
    for (let i = 0; i < 5; i++) {
        webgazer.recordScreenPosition(x, y, 'click');
    }

    pointElement.classList.add('clicked');
    this.currentCalibrationPoint++;

    if (this.currentCalibrationPoint >= 9) {
        this.endCalibration();
    }
}
```

C.4 Gaze Data Export

Listing C.4: Gaze data export

```
exportData() {
    const dataStr = JSON.stringify({
        metadata: {
            exportTime: new Date().toISOString(),
            totalPoints: this.gazeData.length,
            duration: this.gazeData[this.gazeData.length - 1].
                timestamp
                    - this.gazeData[0].timestamp
        },
        gazePoints: this.gazeData
    }, null, 2);

    const blob = new Blob([dataStr], { type: 'application/json' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `eye_tracking_data_${Date.now()}.json`;
    a.click();
}
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

C.5 Eye-Tracking Launch Command

Listing C.5: Eye-tracking launch command

```
1  private handleOpenStandaloneEyeTracking(): void {
2      const standaloneHtmlPath = path.join(
3          this.extensionContext.extensionPath,
4          '..', '..', 'packages', 'eye-tracking',
5          'standalone', 'standalone_eye_tracking_demo.html'
6      );
7
8      if (!fs.existsSync(standaloneHtmlPath)) {
9          vscode.window.showErrorMessage('Eye tracking demo not found')
               ;
10         return;
11     }
12
13     const fileUri = vscode.Uri.file(standaloneHtmlPath);
14     vscode.env.openExternal(fileUri);
15 }
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX D

Interaction Replay Code

This appendix contains the implementation details for the interaction replay system described in Section 4.5.

D.1 ID Pre-Mapping During Event Analysis

Listing D.1: ID pre-mapping during event analysis

```
1  private buildCreationIdMap(events: InteractionEvent[]): void {
2      // Find all createNode events
3      const createNodes: { index: number; elementType: string;
4                           event: InteractionEvent }[] = [];
5      for (let i = 0; i < events.length; i++) {
6          const event = events[i];
7          if (event.type === InteractionEventType.ELEMENT_CREATE
8              && event.data.kind === 'createNode') {
9              createNodes.push({ index: i, elementType: event.data.
                   elementTypeId, event });
10         }
11     }
12
13     // Find first occurrence of each ID (from selections, edges, etc
           .)
14     const idFirstOccurrence: Map<string, number> = new Map();
15     for (let i = 0; i < events.length; i++) {
16         const ids = this.extractIdsFromEvent(events[i]);
17         for (const id of ids) {
18             if (!idFirstOccurrence.has(id)) {
19                 idFirstOccurrence.set(id, i);
20             }
21         }
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    }

    // Assign each ID to the oldest unassigned createNode before it
    const assignedCreateNodes = new Set<number>();
    const sortedIds = Array.from(idFirstOccurrence.entries())
        .sort((a, b) => a[1] - b[1]);

    for (const [id, firstOccurrence] of sortedIds) {
        let bestCreateNode = null;
        for (const cn of createNodes) {
            if (assignedCreateNodes.has(cn.index)) continue;
            if (cn.index >= firstOccurrence) continue;
            if (!bestCreateNode || cn.index < bestCreateNode.index) {
                bestCreateNode = cn;
            }
        }
        if (bestCreateNode) {
            (bestCreateNode.event.data as any)._oldElementId = id;
            assignedCreateNodes.add(bestCreateNode.index);
        }
    }
}
```

D.2 Runtime ID Capture from Server Responses

Listing D.2: Capturing new element IDs from server responses

```
private setupModelListener(): any {
    return this.actionListener.registerServerListener((message: any)
        => {
        const action = message?.action;
        if (action?.kind === 'updateModel' && action.newRoot) {
            this.extractNewElementIds(action.newRoot);
        }
    });
}

private extractNewElementIds(root: any): void {
    const allElements = this.getAllElements(root);

    if (this.pendingElementCreation) {
        const expectedType = this.pendingElementCreation.elementType;

        const matchingElements = allElements.filter(el => {
            const typeMatches = this.elementTypesMatch(el.type,
                expectedType);
            const isNew = !this.knownElementIds.has(el.id);
            return typeMatches && isNew;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
        });

        if (matchingElements.length > 0) {
            const newElement = matchingElements[0];
            this.pendingElementCreation.resolve(newElement.id);
            this.pendingElementCreation = null;
        }
    }

    // Track all known elements
    allElements.forEach(el => {
        if (el.id) this.knownElementIds.add(el.id);
    });
}
```

D.3 ID Translation for Edge Creation

Listing D.3: ID translation for edge creation

```
if (data.kind === 'createEdge') {
    const sourceId = this.idMapping.get(data.sourceElementId)
                   || data.sourceElementId;
    const targetId = this.idMapping.get(data.targetElementId)
                   || data.targetElementId;

    const sourceHasMapping = this.idMapping.has(data.sourceElementId)
        ;
    const targetHasMapping = this.idMapping.has(data.targetElementId)
        ;

    if (sourceHasMapping && targetHasMapping) {
        return {
            kind: 'createEdge',
            elementTypeId: data.elementTypeId,
            sourceElementId: sourceId,
            targetElementId: targetId,
            isOperation: true
        };
    } else {
        console.warn('Skipping edge - missing ID mapping');
        return null;
    }
}
```

D.4 Container Tracking for Element Creation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Listing D.4: Container tracking for element creation

```
private updateContainerTracking(elementType: string, mappedId: string
    ): void {
    if (elementType === 'ACTIVITY__Activity') {
        this.lastSelectedActivityId = mappedId;
    }
    else if (elementType === 'CLASS__Class' ||
             elementType === 'CLASS__Interface') {
        this.lastSelectedClassId = mappedId;
    }
    else if (elementType === 'STATE_MACHINE__Region') {
        this.lastSelectedRegionId = mappedId;
    }
    // Additional container types...
}

private getContainerIdForElementType(elementTypeId: string): string |
     null {
    // Activity nodes need Activity container
    if (ACTIVITY_NODE_TYPES.includes(elementTypeId)) {
        return this.lastSelectedActivityId;
    }
    // State Machine elements need Region container
    if (STATE_MACHINE_REGION_TYPES.includes(elementTypeId)) {
        return this.lastSelectedRegionId;
    }
    // Class members need Class/Interface container
    if (CLASS_MEMBER_TYPES.includes(elementTypeId)) {
        return this.lastSelectedClassId;
    }
    return null;
}
```

D.5 Speed-Controlled Replay

Listing D.5: Speed-controlled replay

```
private async replayEvents(
    events: InteractionEvent[],
    options: ReplayOptions,
    progress: vscode.Progress<{ message?: string; increment?: number
        }>
): Promise<void> {
    const speed = options.speed ?? 1;
    const instantReplay = speed === 0;

    for (let i = 0; i < events.length; i++) {
        if (this.replayAbortController?.signal.aborted) return;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
11
12          const event = events[i];
13          const nextEvent = events[i + 1];
14
15          progress.report({
16              message: `Event␣${i␣+␣1}/${events.length}:␣${event.type}`
                    ,
17              increment: 100 / events.length
18          });
19
20          await this.dispatchEvent(event);
21
22          if (!instantReplay && nextEvent) {
23              const delay = (nextEvent.timestamp - event.timestamp) /
                    speed;
24              await this.sleep(Math.max(delay, 100));
25          }
26      }
27  }
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX E

bigUML Integration Code

This appendix contains the implementation details for the BIGUML integration described in Chapter 5.

E.1 Package Dependency Configuration

Listing E.1: Package dependency configuration

```
{
  "name": "@borkdominik-biguml/big-eye-tracking",
  "version": "0.6.3",
  "dependencies": {
    "@eclipse-glsp/protocol": "^2.2.0",
    "@eclipse-glsp/client": "^2.2.0"
  }
}
```

E.2 Module Registration in VSCode Extension

Listing E.2: Module registration in VSCode extension

```
// In the main extension activation
bind(EYE_TYPES.InteractionTracker).to(InteractionTracker).
    inSingletonScope();
bind(EYE_TYPES.InteractionReplayService).to(InteractionReplayService)
    .inSingletonScope();
bind(EyeTrackingProvider).toSelf().inSingletonScope();
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

E.3 GLSP Client Module for Tracking

Listing E.3: GLSP client module for tracking

```
const eyeTrackingModule: ContainerModule = new ContainerModule(bind
    => {
    // Register viewport tracking handler
    bind(ViewportTrackingHandler).toSelf().inSingletonScope();
    bind(TYPES.IActionHandler).toService(ViewportTrackingHandler);

    // Register element bounds tracking handler
    bind(ElementBoundsTrackingHandler).toSelf().inSingletonScope();
    bind(TYPES.IActionHandler).toService(ElementBoundsTrackingHandler
        );
});

export default eyeTrackingModule;
```

E.4 Eye-Tracking Provider Setup

Listing E.4: Eye-tracking provider setup

```
@injectable()
export class EyeTrackingProvider extends BIGReactWebview {
    @inject(EYE_TYPES.InteractionTracker)
    protected readonly interactionTracker: InteractionTracker;

    protected override handleConnection(): void {
        super.handleConnection();

        this.toDispose.push(
            this.webviewConnector.onActionMessage(message => {
                if (StartTrackingSessionAction.is(message.action)) {
                    this.handleStartTrackingSession(message.action);
                } else if (StopTrackingSessionAction.is(message.
                    action)) {
                    this.handleStopTrackingSession();
                } else if (UploadSessionToServerAction.is(message.
                    action)) {
                    this.handleUploadSessionToServer();
                }
            })
        );
    }
}
```

E.5 Session Upload to Server

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Listing E.5: Session upload to server

```
private async handleUploadSessionToServer(): Promise<void> {
    const fileUris = await vscode.window.showOpenDialog({
        filters: { 'JSON Files': ['json'] },
        title: 'Select Session JSON File to Upload'
    });

    if (!fileUris || fileUris.length === 0) return;

    const fileContent = fs.readFileSync(fileUris[0].fsPath, 'utf-8');
    const sessionData = JSON.parse(fileContent);

    const serverUrl = vscode.workspace.getConfiguration('bigUML')
        .get<string>('monitoringServerUrl') || 'http://localhost:8000
           ';

    const response = await fetch(`${serverUrl}/sessions`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: fileContent
    });

    if (response.ok) {
        const result = await response.json();
        vscode.window.showInformationMessage(
            `Session uploaded! ${result.events_stored} events stored.
               `
        );
    }
}
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

APPENDIX F

Integration Guide for Third-Party Modeling Frontends

This appendix is intended as a practical reference for developers who wish to integrate the monitoring framework presented in Chapter 4 into a modeling frontend other than BIGUML. The framework was deliberately designed to be tool-agnostic at the backend layer: the database schema (Section 4.1.2) uses generic field names such as `tool_id` and `editor_type`, the event row stores its full payload in a JSONB column, and the REST endpoints accept a single canonical JSON envelope regardless of the originating tool. The goal of this appendix is to make those affordances explicit so that a second frontend can be integrated without modifying the backend codebase.

The guide is organized in five parts. Section F.1 describes the wire-level contract that any client must satisfy. Section F.2 clarifies which fields of the schema are interpreted by the backend and which are passed through to the JSONB columns unchanged. Section F.3 gives the recommended mapping from generic tool-native operations to the canonical event types used by the analytical pipelines. Section F.4 sketches a minimal implementation recipe. Section F.5 covers the optional eye-tracking integration. Section F.6 closes with a validation checklist.

F.1 The Backend Contract

From a client's point of view the backend is a small set of REST endpoints served by FastAPI (Section 4.1.2). The minimal surface a frontend must use is a single `POST` for the session, plus optionally a `POST` for the corresponding gaze stream. Everything else (retrieval, statistics, XES export, friction analysis, heatmap generation) is read-only and consumed by analysis tooling rather than by the frontend.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Base URL and authentication. The base URL is configurable per deployment (the BIGUML integration exposes it as a VSCode setting, see Appendix E.5). All write endpoints require an API key supplied via an HTTP header (if the backend is configured for a production environment, during development it can be left empty). The key is validated through a FastAPI dependency injection chain that keeps authentication logic separate from business logic, as documented in Section 4.1.2. A client that fails to supply a valid key receives an `HTTP 401` response.

Core endpoints used by a frontend. Table F.1 summarises the endpoints relevant for integration. Endpoints not listed (statistics, XES export, heatmap, friction analysis) are not called by the frontend and therefore do not constrain the integration.

Table F.1: Endpoints relevant to a frontend integration.

Method	Path	Purpose
`POST`	`/sessions`	Upload a complete interaction session (envelope + events). Returns the assigned session UUID and the number of events stored.
`POST`	`/sessions/{session_id}/gaze`	Append a gaze-point stream to an existing session (optional, only if eye-tracking is enabled).
`DELETE`	`/sessions/{session_id}`	Permanently remove a session together with its events and gaze points. Used for GDPR-compliant deletion (Section 4.1.2).

Atomic uploads. The framework deliberately does not expose a per-event streaming endpoint. The primary endpoint accepts a complete recording session as a single JSON payload, which simplifies the client implementation and ensures data consistency, as sessions are stored atomically (Section 4.1.2). A frontend should therefore buffer events locally for the duration of a session and upload the complete envelope on session end. This also matches the local-storage-by-default principle laid out in Section 5.6: data lives on the user's machine first, and is forwarded to the server only on explicit user action.

F.2 Fixed Fields vs Flexible JSONB

The schema combines two complementary mechanisms. A small set of *fixed* fields is parsed by the backend, indexed in PostgreSQL, and consumed by the analytical pipelines (Section 4.2). A larger set of *flexible* fields is stored in JSONB columns and is available for any tool-specific data the frontend wishes to retain. This section distinguishes the two so that an integrator knows exactly where their tool-specific concepts may live.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

F.2.1 Upload Envelope

The upload payload is the JSON document shown in Listing F.1. Two top-level keys are required: `session` (an object describing the recording) and `events` (an ordered array of interaction events).

Listing F.1: Canonical upload envelope (`POST /sessions`).

```
{
  "session": {
    "sessionId":   "my-tool-2026-05-10T14-23-08-abc123",
    "startTime":   "2026-05-10T14:23:08.000Z",
    "endTime":     "2026-05-10T14:29:42.000Z",
    "toolId":      "MyModelingTool",
    "toolVersion": "1.4.2",
    "editorType":  "bpmn",
    "modelFile":   "order-process.bpmn",
    "user":        "anon-7f3c",
    "workspace":   "study-2026",
    "extraData":   { "studyArm": "A", "screen": { "w": 1920, "h": 1080 } }
  },
  "events": [
    {
      "timestamp": "2026-05-10T14:23:09.117Z",
      "type":      "element_create",
      "data": {
        "kind":          "createNode",
        "elementTypeId": "bpmn:Task",
        "containerId":   "$ROOT",
        "location":      { "x": 240, "y": 160 }
      }
    },
    {
      "timestamp": "2026-05-10T14:23:11.443Z",
      "type":      "element_edit",
      "data": {
        "kind":     "applyLabelEdit",
        "labelId":  "node-uuid-001-label",
        "text":     "Receive Order"
      }
    }
  ]
}
```

Fixed session fields. The following fields are parsed and stored in their dedicated columns of the `recording_sessions` table (Appendix A):
`sessionId`, `startTime`, `endTime`, `toolId`,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

toolVersion, editorType, modelFile, modelFilePath, user, and workspace. Among these only sessionId and startTime are strictly required; the others default to NULL (or 'unknown' for toolId) if the client omits them, but populating them is strongly recommended because the retrieval and statistics endpoints use them as filter criteria. Timestamps must be ISO 8601 strings.

Flexible session metadata. Any additional tool-specific session-level information should be placed under extraData. The backend stores this object in the extra_data JSONB column without inspecting its contents. Use this for experimental-condition tags, screen geometry, build hashes or feature-flag information. The same field can also hold a screenshot path or a study identifier that the client wishes to surface on the retrieval endpoint.

F.2.2 Event Schema

Each entry of the events array becomes one row in the interaction_events table. The backend extracts and denormalises key information from the event payload to enable efficient querying (Section 4.1.2); everything else is retained in the data JSONB column. Table F.2 summarises the contract.

Table F.2: Event schema: which fields are parsed vs. stored verbatim.

Field	Handling	Notes
timestamp	Fixed, indexed	ISO 8601 string. Required. Ordering relies on this.
type	Fixed, indexed	Mapped to event_type. See Section F.3.
data.kind	Fixed, indexed	Optional finer-grained sub-type (e.g. createNode, undo). Mapped to event_kind.
data.elementId	Denormalised	Used for element-level filtering and joins.
data.elementTypeId	Denormalised	Used by clickstream granularity and friction hotspots.
data.location.x, data.location.y	Denormalised	Mapped to position_x / position_y. Required for rage-click detection (Section 2.4.4).
data.x, data.y	Denormalised, fallback	Used when location is absent (e.g. raw mouse_click).
data (rest)	JSONB, verbatim	Anything not listed above is preserved unchanged. This is where tool-specific concepts live.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

What the backend will look for. The extraction logic for the `data` payload is intentionally lenient. It explicitly handles several common GLSP action kinds, including node and edge creation, label editing, bounds changes, element selection, and viewport changes. For these actions, the corresponding GLSP kinds are `createNode`, `createEdge`, `applyLabelEdit`, `changeBounds`, `elementSelected`, and `setViewport`. The backend additionally provides a fallback that attempts to extract both the `elementId`, `elementTypeId`, as well as position information from standard field names regardless of the action kind (Section 4.1.2).

What the backend will ignore. Anything in `data` that does not match one of the extracted names is left untouched in the JSONB column. This is the primary extension mechanism for tool-specific information: a BPMN editor might add a `flowKind`, a SysML editor might add a `stereotype`, a domain-specific editor might add a `layerId`. None of these require backend changes; they will appear in the JSON exports, in the replay logs, and in the raw event rows, and any downstream analyzer is free to read them.

F.2.3 Gaze Stream

Gaze data uses the same envelope-plus-array shape on a separate endpoint, shown in Listing F.2. Because gaze data is high-frequency (typically tens of samples per second), it is kept in a dedicated `gaze_points` table optimized for simple inserts and time-based queries (Section 4.1.2).

Listing F.2: Gaze upload payload (`POST /sessions/{session_id}/gaze`).

```
{
  "gazePoints": [
    { "x": 731.2, "y": 412.5, "timestamp": "2026-05-10T14:23:09.045Z"
        },
    { "x": 728.8, "y": 410.1, "timestamp": "2026-05-10T14:23:09.078Z"
        }
  ],
  "metadata": {
    "tracker":          "WebGazer",
    "trackerVersion":   "2.0.1",
    "screenWidth":      1920,
    "screenHeight":     1080
  }
}
```

The only fixed fields per gaze point are the screen coordinates `x` and `y` and the `timestamp`. Optional fields `xNormalized`, `yNormalized`, `pupilDiameter`, `confidence`, and `eye` are recognised and stored in their dedicated columns when present, and the entire point object is also retained in the `raw_data` JSONB column. Tracker-specific fields go there without any backend change. The `metadata` block at the top level is preserved on the session and is mainly used by the heatmap endpoint to resolve coordinate scaling.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

F.3 Mapping Tool-Native Operations to Canonical Event Types

The analytical pipelines do not care what GUI gesture produced an event; they care about the *type* of the event and, optionally, its *kind* and *element type*. The clickstream analyzer (Section 2.4.1) constructs event labels of the form `event_type` as well as `event_type:event_kind`, or `event_type:event_kind:element_type` depending on the granularity setting; the friction detector (Section 2.4.4) recognises specific kinds (`undo`, `redo`, `setViewport`, `deleteElement`); the XES exporter (Section 4.1.2) filters out `mouse_click` and `element_select` as noise. Mapping a new frontend onto these conventions therefore boils down to choosing the right `type` and `kind` values.

Table F.3: Canonical `event_type` values and analytical treatment.

type	**Semantic**	**Pipelines**
`element_select`	Selection of one or more model elements.	Excluded from XES
`element_create`	Creation of a node or edge (use `data.kind` `createNode` or `createEdge`).	Clickstream, XES
`element_delete`	Deletion of a model element.	Friction, Clickstream, XES
`element_move`	Bounds change with position delta.	Clickstream, XES
`element_resize`	Bounds change with size delta only.	Clickstream, XES
`element_edit`	Compound or label edit (use `data.kind` `applyLabelEdit`, `compound`, etc.).	Clickstream, XES
`property_change`	Property-palette modification.	Clickstream, XES
`viewport_change`	Pan and/or zoom; emit only after debouncing.	Friction, Clickstream
`zoom_change`	Optional dedicated zoom event if separated from pan.	Friction, Clickstream
`scroll`	Scroll within the editor canvas or its containers.	Friction
`text_edit`	Free-text edit not bound to a label.	Clickstream, XES
`mouse_click`	Raw pointer click (used by the rage-click detector).	Friction; excluded from XES
`session_start`	First event of the session.	Metadata only
`session_end`	Last event of the session.	Metadata only

Canonical event types. Table F.3 lists the canonical `type` values established by the reference implementation (Appendix B.2) together with their intended semantics and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

how the analytical pipelines treat them. A third-party frontend should reuse these values whenever possible; if a new type is genuinely necessary it can be added freely, but it will be ignored by the friction detector and excluded by default from the XES export until the pipelines are updated to recognise it.

Choosing `kind`. The `data.kind` field provides the second granularity level for the clickstream analyzer and is the primary discriminator for the friction detector. The detector recognises `undo` / `UndoAction` and `redo` / `RedoAction` for undo/redo bursts (matching either the top-level field or the nested `data.kind` field, see Section 2.4.4); `setViewport`, `center`, `fit`, `scroll`, and `zoom` for navigation thrashing; and `deleteElement` for rapid-deletion bursts. Any tool-native operation that semantically corresponds to one of these should reuse the exact string so that the existing detection logic fires without modification.

Choosing the element-type identifier. The element-type identifier (`data.elementTypeId`) is the third granularity level for the clickstream analyzer. BIGUML uses double-underscore-prefixed identifiers such as `CLASS__Class` and `ACTIVITY__OpaqueAction` (Section 5.6); the analyzer's `_clean_element_type` helper strips the namespace prefix so the exported labels are readable. Other tools may use different conventions (e.g. qualified BPMN names like `bpmn:Task`) and these will appear unchanged in the exported labels. If a tool prefers a different cleaning rule, the `ClickstreamAnalyzer` constructor exposes the granularity parameter so a custom prefix can be stripped without changing the database.

F.4 Implementation Recipe

The following recipe distils the design of the BIGUML frontend library (Section 4.3) into the minimal set of steps that any modeling frontend must perform. It is deliberately framework-neutral; the example code in Listing F.3 is plain TypeScript and makes no GLSP-specific assumptions.

1. Session lifecycle. Maintain a single `Session` object that tracks the current session identifier, the start timestamp, and any tool-level metadata. Start the session on explicit user opt-in; record a `session_start` event; buffer subsequent events in memory; record a `session_end` event when the user stops tracking or the editor closes; serialize the result to JSON and either write it to disk (local-first, recommended) or upload it immediately.

2. Event capture. For each operation on the editor canvas, emit one event with the `type` taken from Table F.3 and a `data` object that contains the canonical recognized fields (`kind`, `elementId`, `elementTypeId`, `location`) wherever applicable. Any further detail relevant to the tool can be added to `data` freely. Use the editor's command

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

bus, action listener, or model-change subscription as the capture point rather than DOM events directly; this is what gives the resulting log its *semantic* character.

3. Throttling. High-frequency events (continuous pan, zoom, drag) must be throttled or debounced before they are appended to the event buffer. The reference implementation uses a debounce timeout of 100 ms for viewport changes (Appendix B.5); other frontends should adopt a comparable strategy. Skipping this step is the single most common cause of high overhead during integration (Section 5.6).

4. Local persistence then optional upload. The reference implementation writes one JSON file and one CSV file per session before uploading (Appendix B.8). A third-party integration is free to skip the CSV output, but the local JSON file is strongly recommended: it gives the user a transparent artifact to review (consistent with the documentation requirement in Section 5.6) and acts as a recovery path when the network upload fails.

5. Upload. On user-initiated upload, `POST` the envelope to `/sessions` with the API key in the configured header. The response body contains the backend-assigned session UUID and the event count; the frontend should surface both to the user so they can verify the upload succeeded.

Listing F.3: Minimal tool-agnostic client (excerpts).

```
type CanonicalEventType =
    | 'element_select' | 'element_create' | 'element_delete'
    | 'element_move'   | 'element_resize' | 'element_edit'
    | 'property_change' | 'viewport_change' | 'zoom_change'
    | 'scroll' | 'file_open' | 'file_save' | 'file_close'
    | 'text_edit' | 'mouse_click'
    | 'session_start' | 'session_end';

interface InteractionEvent {
    timestamp: string;              // ISO 8601
    type:      CanonicalEventType;
    data:      Record<string, unknown>;  // recognised + tool-
        specific
}

interface SessionEnvelope {
    sessionId:    string;
    startTime:    string;           // ISO 8601
    endTime?:     string;
    toolId:       string;           // e.g. "MyModelingTool"
    toolVersion?: string;
    editorType?:  string;           // e.g. "bpmn", "class", "feature-
        model"
    modelFile?:   string;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
23      user?:        string;
24      workspace?:   string;
25      extraData?:   Record<string, unknown>;
26  }
27
28  class MonitoringClient {
29      private events: InteractionEvent[] = [];
30      private session?: SessionEnvelope;
31
32      start(envelope: SessionEnvelope): void {
33          this.session = envelope;
34          this.track('session_start', {});
35      }
36
37      track(type: CanonicalEventType, data: Record<string, unknown>):
            void {
38          if (!this.session) return;
39          this.events.push({
40              timestamp: new Date().toISOString(),
41              type,
42              data
43          });
44      }
45
46      async stopAndUpload(serverUrl: string, apiKey: string): Promise<
            void> {
47          if (!this.session) return;
48          this.track('session_end', {});
49          this.session.endTime = new Date().toISOString();
50
51          const body = JSON.stringify({
52              session: this.session,
53              events:  this.events
54          });
55
56          const response = await fetch(`${serverUrl}/sessions`, {
57              method:  'POST',
58              headers: {
59                  'Content-Type':  'application/json',
60                  'X-API-Key':     apiKey
61              },
62              body
63          });
64
65          if (!response.ok) {
66              throw new Error(`Upload failed: ${response.status}`);
67          }
68      }
69  }
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6. Wiring to the editor. The remaining work is editor-specific: connecting the `track()` call to the editor's command, action, or model-change subscription. For a GLSP client the natural hook is the action listener (Appendix B.3); for a Sirius client it would be the EMF notification chain (similar to MER, Section 2.2.2); for a custom canvas it is whatever event channel the canvas exposes. The mapping in Table F.3 stays the same regardless.

F.5 Optional: Eye-Tracking Integration

Eye-tracking is intentionally decoupled from the main tracking library (Section 4.4). The reference implementation runs gaze capture in a standalone browser window using WebGazer and joins the resulting stream to the corresponding session by sharing the session identifier. A third-party frontend can either reuse the standalone application unchanged or substitute a different tracker.

Reusing the existing standalone application. The standalone HTML file shipped with the BIGUML integration (Appendix C.1) requires no modification to work with a different host tool. The host tool only needs to surface a button that opens the standalone application as an external browser tab, and a mechanism for the user to enter the session identifier so the gaze stream can later be matched on the backend.

Substituting a different tracker. Because the gaze upload endpoint accepts the generic format shown in Listing F.2, any tracker that can produce $\langle x, y, t \rangle$ samples can be plugged in. The `metadata` block at the top of the payload should at minimum identify the tracker and the screen geometry; additional tracker-specific fields can live either at the gaze-point level (where they are preserved in `raw_data`) or in `metadata`. The Gaussian-smoothed heatmap generation in Section 2.4.3 works on the screen coordinates only and is therefore tracker-agnostic; the sigma parameter is exposed on the heatmap endpoint so it can be tuned to the accuracy of the chosen tracker.

Linking gaze to a session. The backend joins gaze and interaction streams on the session identifier that appears in both uploads. The interaction session must therefore exist before the gaze upload, or be uploaded in the same transaction; the BIGUML integration takes the former approach and uploads the interaction session first.

F.6 Validation Checklist

The following checklist captures the operational steps that proved useful during the BIGUML integration and that a new integration can follow to verify correctness end-to-end.

1. **Envelope round-trip.** Upload a minimal session containing only a `session_start` and a `session_end` event with no interaction in between. Verify that the `POST`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

`/sessions` response reports `events_stored = 2` and that the session is retrievable via the corresponding `GET` endpoint.

2. **Type denormalisation.** Upload a session containing one `element_create` event with `data.kind = "createNode"`, `data.elementTypeId`, and a `data.location`. Query the database (or the retrieval endpoint) and confirm that `event_type`, `event_kind`, `element_type`, `position_x`, and `position_y` are populated.

3. **JSONB preservation.** Add a tool-specific field to the `data` object (e.g. `"layerId": "L2"`). Confirm that the field survives round-trip retrieval unchanged – this is the primary indicator that the extension mechanism works as intended.

4. **Throttling.** Drag an element continuously for ten seconds and inspect the resulting event count. If the buffer holds more than a few dozen `element_move` or `viewport_change` events for that interval, the debouncer is not active.

5. **Clickstream label.** Run the clickstream analyzer at granularity `full` against the uploaded session. The labels in the n-gram output should read as `event_type:event_kind:element_type`, which confirms that the mapping in Section F.3 has been applied consistently.

6. **Friction integration.** Trigger an undo-redo sequence in the editor and confirm that the friction analysis endpoint reports a non-zero undo/redo sub-score for the uploaded session. If the score is zero, the `data.kind` values likely do not match the strings expected by the detector.

7. **Gaze join (if applicable).** Upload a small gaze stream bound to an existing session identifier. Request the heatmap endpoint for that session and verify that a PNG response is returned and, where a screenshot is available, that the overlay is rendered.

8. **Deletion.** Issue `DELETE /sessions/{id}` for the test session. Confirm that subsequent retrieval returns `HTTP 404` and that the cascading delete also removes the associated events and gaze points.

A frontend that passes the eight checks above is correctly integrated for all functionality currently exercised by the framework. Anything beyond that – extra analytics, custom visualisations, automated upload – is layered on top of the same wire format and requires no further backend changes.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

4.1 Database Schema 33
4.2 Swagger documentation of the backend REST-API 35

6.1 Reference UML class diagram used in the user study, representing a simple university scenario with an inheritance hierarchy and two labeled associations. 58
6.2 Aggregate gaze heatmap for participant P2 (property-centric, 351 s) overlaid on a screenshot of the BIGUML editor. Warmer colors indicate higher gaze density. 64
6.3 Aggregate gaze heatmap for participant P3 (layout-centric, 247 s) overlaid on a screenshot of the BIGUML editor. 65
6.4 Aggregate gaze heatmap for participant P7 (layout-centric, 291 s, longest event-richest session) overlaid on a screenshot of the BIGUML editor. The strongest concentrations follow the inheritance hierarchy and the path of the *Course* associations. 65
6.5 Participant sessions P2 and P3 imported into ProM from the framework's XES export. Each session is represented as a trace containing the chronologically ordered semantic modeling events. 67
6.6 Detailed excerpt from participant P2's trace in ProM. The sequence shows the creation of an abstract class, which was not part of the reference model, and its subsequent deletion. 67
6.7 Process model generated by Disco from participant P7's exported XES session. Nodes represent recorded semantic modeling activities and edges represent observed transitions between consecutive activities. 68

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

2.1 Comparative overview of existing monitoring tools and frameworks. . . . 16
2.2 Interaction Types, Events, Tracking Methodologies, and Interpretation Potential 17

3.1 Summary of functional requirements. 29
3.2 Summary of non-functional requirements. 30
3.3 Requirement for the reference integration. 30

5.1 Identifier fields used by different action kinds. 53

6.1 Per-session summary of duration, captured events, dominant modeling strategy, composite friction score (0–100), the number of detected rage-click incidents (RC ev.), and the total number of rapid clicks they contained (RC clicks). No undo/redo bursts and no rapid-deletion bursts were detected in any session; navigation thrashing was detected only in P3 (1 incident). 60
6.2 Evaluation of the framework against the requirements defined in Chapter 3. 71

F.1 Endpoints relevant to a frontend integration. 106
F.2 Event schema: which fields are parsed vs. stored verbatim. 108
F.3 Canonical `event_type` values and analytical treatment. 110

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[Aal16] Wil Aalst. *Process Mining: Data Science in Action*. Springer, 3 edition, January 2016.

[AATWL25] Amine Abbad-Andaloussi, Dung My Thi Trinh, Barbara Weber, and Hugo A. López. On the relationship between semantic transparency and cognitive load: Does context matter in process models? In Fred D. Davis, René Riedl, Jan vom Brocke, Pierre-Majorique Léger, Adriane B. Randolph, and Gernot R. Müller-Putz, editors, *Information systems and neuroscience*, pages 193–203, Cham, 2025. Springer Nature Switzerland.

[AEN20] Gunes Acar, Steven Englehardt, and Arvind Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. *Proceedings on Privacy Enhancing Technologies*, 2020(4):220–238, October 2020.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, 1995. Conference Name: Eleventh International Conference on Data Engineering ISBN: 9780818669101 Place: Taipei, Taiwan.

[ASJW09] David Akers, Matthew Simpson, Robin Jeffries, and Terry Winograd. Undo and erase events as indicators of usability problems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 659–668, New York, NY, USA, April 2009. Association for Computing Machinery.

[BA22] Abednego Bansah and Douglas Agyei. Perceived convenience, usefulness, effectiveness and user acceptance of information technology: evaluating students' experiences of a Learning Management System. *Technology, Pedagogy and Education*, 31:431–449, January 2022.

[BLO23] Dominik Bork, Philip Langer, and Tobias Ortmayr. A vision for flexible glsp-based web modeling tools. In *The Practice of Enterprise Modeling: 16th IFIP Working Conference, PoEM 2023, Vienna, Austria, November 28–December 1, 2023, Proceedings*, volume 497, pages 109–124, Cham, 2023. Springer.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Boj13] Agnieszka Bojko. *Eye Tracking the User Experience: A Practical Guide to Research*. December 2013.

[BS14] Jennifer Romano Bergstrom and Andrew Schall. *Eye Tracking in User Experience Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2014.

[BTI07] David Borland and Russell M. Taylor Ii. Rainbow Color Map (Still) Considered Harmful. *IEEE Computer Graphics and Applications*, 27(2):14–17, March 2007.

[BvZA19] Alessandro Berti, Sebastiaan van Zelst, and Wil Aalst. Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science. May 2019.

[Cav09] Ann Cavoukian. Privacy by Design The 7 Foundational Principles. 2009.

[CLB+03] Irina Ceaparu, Jonathan Lazar, Katie Bessiere, John Robinson, and Ben Shneiderman. Determining Causes and Severity of End-User Frustration (2002). *International Journal of Human-Computer Interaction*, 17, February 2003.

[Cli26] ClickHouse. ClickHouse, 2026. https://clickhouse.com/docs [Accessed 2026-03-31] Citation Key: ClickHouse2026.

[CS08] Gay Costain and Ananth Srinivasan. A Modeling Methodology for Empirically Studying User Behavior: The Case of UML Diagram Usage. In *Proceedings of the 13th International Workshop on Exploring Modeling Methods for Systems Analysis and Design, EMMSAD 2008, held in conjunction with the CAiSE 2008 Conference, Montpellier, France, June 16-17, 2008*, CEUR Workshop Proceedings, pages 55–70. CEUR-WS.org, 2008.

[DBW23] MohammadHadi Dehghani, Luca Berardinelli, and Manuel Wimmer. Towards Modeling Process Mining for Graphical Editors. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 929–933, Västerås, Sweden, October 2023. IEEE.

[Duc17] Andrew T. Duchowski. *Eye Tracking Methodology*. Springer, London, 3 edition, 2017.

[FLDAGRT22] Daniel Fernández-Lanvin, Javier De Andrés, Martín González-Rodríguez, and Pelayo Torre. Interaction Lab: Web User Interaction Tracking and Analysis Tool:. In *Proceedings of the 18th International Conference on Web Information Systems and Technologies*, pages 152–158, Valletta, Malta, 2022. SCITEPRESS - Science and Technology Publications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[Ful22] Fullstory. What are Rage Clicks? How to Identify Frustrated Users | Fullstory, 2022. https://www.fullstory.com/blog/rage-clicks/ [Accessed 2026-01-27].

[Ful26] Fullstory. FullStory, 2026. https://www.fullstory.com [Accessed 2026-03-24] Citation Key: FullStory2026.

[GMW22] Frances Grodzinsky, Keith Miller, and Marty Wolf. Session Replay Scripts: A Privacy Analysis. *School of Computer Science & Engineering Faculty Publications*, January 2022.

[GOH+23] Céline Gressel, Rebekah Overdorf, Inken Hagenstedt, Murat Karaboga, Helmut Lurtz, Michael Raschke, and Andreas Bulling. Privacy-Aware Eye Tracking: Challenges and Future Directions. *IEEE Pervasive Computing*, 22:95–102, March 2023.

[Gra26] Dominik Granig. Interaction tracking dataset for web-based modeling tools. `https://doi.org/10.5281/zenodo.20112314`, 2026. Last accessed 2026-08-10.

[HMJR04] Alan Hevner, Salvatore March, Park Jinsoo, and Sudha Ram. Design Science in Information Systems Research. *Management Information Systems Quarterly*, 28:75, March 2004.

[HNA+11] Kenneth Holmqvist, Marcus Nyström, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, and Joost van de Weijer. *Eye Tracking: A comprehensive guide to methods and measures*. Oxford University Press, Oxford, New York, 2011.

[Hot26] Hotjar. Hotjar: Website Heatmaps & Behavior Analytics Tools, 2026. https://www.hotjar.com/ [Accessed 2026-04-05] Citation Key: Hotjar2026.

[IEE16] IEEE. 1849-2016 - IEEE Standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams, 2016. https://upcommons.upc.edu/entities/publication/37265514-d72e-4205-b772-7292828dbc47 [Accessed 2026-01-27].

[Kha25] Linh Khanh. User Behavior Tracking: Methods and Best Tools, March 2025.

[KL22] Reyhaneh Kalantari and Timothy C. Lethbridge. Characterizing UX Evaluation in Software Modeling Tools: A Literature Review. *IEEE Access*, 10:131509–131527, 2022.

[KLM20] Jacob Kröger, Otto Lutz, and Florian Müller. What Does Your Gaze Reveal About You? On the Privacy Implications of Eye Tracking. In

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Privacy and Identity Management. Data for Better Living: AI and Privacy, pages 226–241. March 2020.

[KMA04] Andrew Ko, Brad Myers, and Htet Aung. Six Learning Barriers in End-User Programming Systems. In *2004 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2004), 26-29 September 2004, Rome, Italy*, pages 199–206, 2004. Pages: 206.

[LS16] James Lewis and Jeff Sauro. *Quantifying the User Experience*. 2 edition, August 2016.

[MB23] Haydar Metin and Dominik Bork. Introducing bigUML: A flexible open-source glsp-based web modeling tool for uml. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 40–44. IEEE, 2023.

[MB25] Haydar Metin and Dominik Bork. A reference architecture for the development of GLSP-based web modeling tools. *Softw. Syst. Model.*, 24(6):1869–1895, 2025.

[MDCSW25] Silvia Miksch, Claudio Di Ciccio, Pnina Soffer, and Barbara Weber. Visual analytics meets process mining: Challenges and opportunities. *IEEE Computer Graphics and Applications*, 44(6):132–141, 2025.

[Nie93] Jakob Nielsen. *Usability Engineering*. Elsevier S&T, 1993. ISBN: 9780125184069.

[NP09] Jakob Nielsen and Kara Pernice. *Eyetracking Web Usability*. New Riders, USA, 1 edition, 2009.

[otEU16] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council, April 2016.

[PC99] Peter Pirolli and Stuart Card. Information Foraging. *Psychological Review*, 106:643–675, October 1999.

[PHCJ25] Jiseung Pyun, Min Heo, Yehui Choe, and Jongwook Jeong. WebTrace: An enhanced system for remote user interaction tracking and analysis in web applications. *SoftwareX*, 30:102075, May 2025.

[Pos26] PostHog. PostHog – We make dev tools for product engineers, 2026. https://posthog.com/ [Accessed 2026-03-31].

[Pri] Secure Privacy. Privacy by Design and Default: Protecting User Data Smartly. https://secureprivacy.ai/blog/privacy-by-design-and-default-comprehensive-guide [Accessed 2026-01-27].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[Pri26] Secure Privacy. Mastering privacy by design | A comprehensive guide to proactive data protection, 2026. https://secureprivacy.ai/de/blog/privacy-by-design-implementation [Accessed: 2026-01-27].

[PS23] Libor Polčák and Alexandra Slezáková. Data Exfiltration by Hotjar Revisited:. In *Proceedings of the 19th International Conference on Web Information Systems and Technologies*, pages 347–354, Rome, Italy, 2023. SCITEPRESS - Science and Technology Publications.

[RTS20] Kristina Rosenthal, Benjamin Ternes, and Stefan Strecker. Understanding individual processes of conceptual modeling: A multi-modal observation and data generation approach. In *Modellierung 2020, Vienna, Austria, February 19–21, 2020, pp. 77–92.*, pages 77–92. Gesellschaft für Informatik e.V., 2020.

[SAAKW25] Thierry Sorg, Amine Abbad-Andaloussi, Ekkart Kindler, and Barbara Weber. Process Model Complexity Metrics, Cognitive Load and Visual Behavior: A Multi-granular Eye-Tracking Analysis. In Renata Guizzardi, Luise Pufahl, Arnon Sturm, and Han van der Aa, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 87–103, Cham, 2025. Springer Nature Switzerland.

[SCDT00] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. *SIGKDD Explorations*, 1:12–23, January 2000.

[TA13] Tom Tullis and Bill Albert. *Measuring the User Experience (Second Edition)*. Interactive Technologies. Morgan Kaufmann, 2013. ISBN: 9780124157811.

[Ter17] Benjamin Ternes. Design and evaluation of a web-based modeling platform to support the learning of conceptual modeling and of studying the corresponding learning processes. *Proceedings of 8th International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA) co-located with the 29th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 138–142, 2017.

[TRBS20] Benjamin Ternes, Kristina Rosenthal, Hagen Barth, and Stefan Strecker. TOOL - Modeling Observatory & Tool: An Update. In *Modellierung*, CEUR Workshop Proceedings, pages 198–202. CEUR-WS.org, 2020.

[TRSB20] Benjamin Ternes, Kristina Rosenthal, Stefan Strecker, and Julian Bartels. TOOL–A Modeling Observatory & Tool for Studying Individual Modeling Processes. In *Demo Track at the 39th International Conference on Conceptual Modeling, ER 2020, TU Wien, Vienna, Austria, Nov 3–6, 2020*, 2020.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[TS18] Benjamin Ternes and Stefan Strecker. A web-based modeling tool for studying the learning of conceptual modeling. pages 325–328. Gesellschaft für Informatik e.V., 2018.

[ZZVW25] L. Zimmermann, Francesca Zerbato, Katerina Vrotsou, and B. Weber. A Process-Oriented Approach to Analyze Analysts' Use of Visualizations: Revealing Insights into the What, When, and How. *Computer Graphics Forum*, 44, May 2025.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.