Author pre-print version. Paper accepted for Tools and Demonstrators at PoEM'26.
Final Paper to be published by CEUR.

How Modelers Create Models: An Open, Tool-Agnostic Framework for Interaction Analytics in Web-Based Modeling Tools

Dominik Granig[1], Charlotte Verbruggen[1] and Dominik Bork[1,*]

[1]TU Wien, Business Informatics Group, Erzherzog-Johann-Platz 1, 1040 Vienna, Austria

Abstract

Web-based modeling tools have become increasingly prominent, 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 tool builders without actionable data to guide usability improvements or to decide whether a new feature is actually adopted. This paper presents an open-source, modular interaction-tracking framework for browser-based modeling tools that captures interactions at both the low-level input and the high-level semantic-action layers and integrates a standalone eye-tracking component. It consists of a TypeScript frontend library, a WebGazer-based eye-tracking module, a Python/FastAPI backend, and a PostgreSQL schema optimized for timestamped event streams; three analytical pipelines provide n-gram clickstream analysis with XES export, a composite friction score, and Gaussian-smoothed gaze heatmaps, and a replay component reconstructs a model step by step from its event log. The framework was integrated into the BIGUML editor, built on the Graphical Language Server Platform (GLSP), and used in an exploratory study with seven participants, which revealed two qualitatively distinct modeling strategies and a concrete, localized usability defect. The dataset is published on Zenodo and the code on GitHub.

Keywords

Interaction Tracking, User Experience, Web-based Modeling Tools, Eye Tracking, Clickstream Analysis, GLSP

1. Introduction

Web-based modeling tools have made modeling more accessible by lowering technical barriers and enabling modeling directly in the browser. Frameworks such as the Graphical Language Server Platform (GLSP)[1], while using standard web technologies, enable developers to build and expand ever-growing ecosystems of modeling tools packed with innovative features [1]. However, the user experience (UX) of modeling tools still remains to be explored in detail [2]. Frameworks such as GLSP focus on infrastructure for implementing graphical editors, while dedicated facilities for recording and analyzing modeler interactions are typically provided by specialized research tools such as TOOL or MER [3, 4, 5].

Newly introduced features are not necessarily adopted or used as intended, and discoverability and perceived value significantly influence users' adoption decisions [6]. Without concrete usage data, it is difficult for tool builders to tell whether limited adoption results from usability issues, insufficient discoverability, or a mismatch between user expectations and functionality. Capturing how users interact with the tool during modeling (i.e., creating, deleting, dragging and resizing elements, and especially where they look) could help tool providers improve usability and contribute towards more user-centric and model-based software engineering [7]. Behavioral analytics frameworks such as PostHog [8], FullStory [9], or Hotjar [10] are either proprietary, hard to configure, or simply not designed for the semantic user actions of modeling environments: they treat the canvas as an unstructured DOM and cannot answer how modelers actually construct a model.

Companion Proceedings of the 19th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling: PoEM Forum, Doctoral Consortium, Business Case & Tool Forum, Workshops (PoEM-C 2026) co-located with PoEM 2026, Mytilene, Greece, December 1-3, 2026

*Corresponding author.

domi.granig@gmail.com (D. Granig); charlotte.verbruggen@tuwien.ac.at (C. Verbruggen); dominik.bork@tuwien.ac.at (D. Bork)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]https://eclipse.dev/glsp/

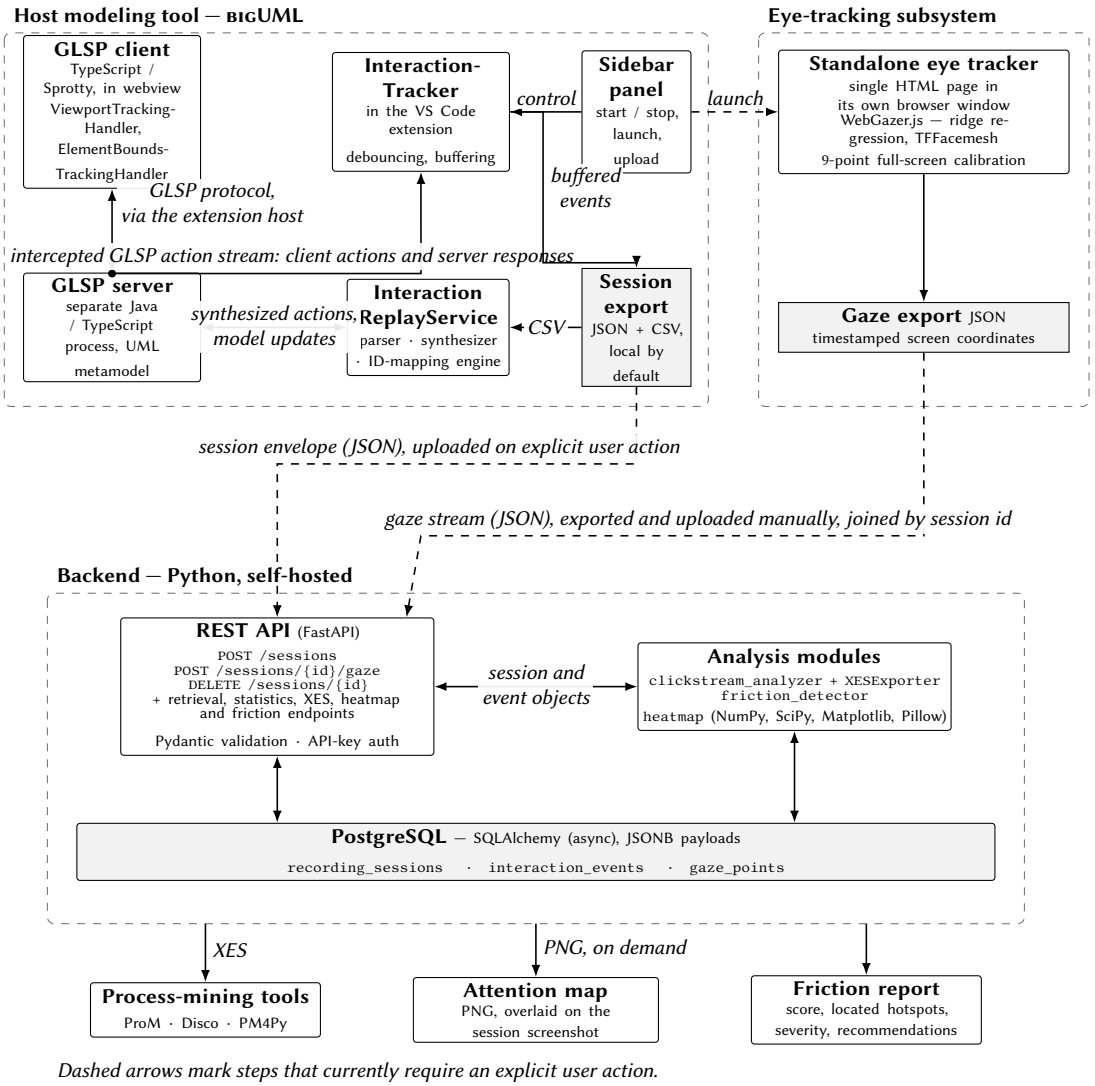


Figure 1: System architecture of the framework for capturing and analyzing modeler-tool interactions.

Contribution. This paper presents an open-source, modular, and tool-agnostic framework for capturing and analyzing user interactions in browser-based modeling tools. Its four artifacts (see Figure 1) are 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. Three analytical pipelines operate on the captured data: n-gram clickstream analysis with IEEE 1849-2016 XES export, a four-component composite friction score, and Gaussian-smoothed gaze heatmaps. A replay component reconstructs a recorded session step by step from its event log. The backend and eye-tracking subsystem can be deployed with any modeling tool, since it is hosted on an independent server and the eye-tracking subsystem runs in its own browser window. A GLSP-based modeling tool is only required to enable the replay component. The framework addresses three questions: i) which interactions are relevant to capture in modeling tools; ii) how the collected data can be structured and analyzed; and iii) how eye-tracking can be embedded in web-based modeling tools. It was integrated into the bigUML editor [11] as a reference integration and exercised in an exploratory study with seven participants. All code is available at[2], the instrumented bigUML fork at[3], a demonstration video at[4], and the raw dataset at[5].

[2]https://github.com/granigd/InteractionMonitoring
[3]https://github.com/granigd/bigUML
[4]https://www.youtube.com/watch?v=DRvEqosXkcY
[5]https://doi.org/10.5281/zenodo.20112314 [12]

2. Background and Related Work

Previous work has analyzed modeling styles and strategies in process modeling [13, 14]. Based on process mining and eye movement analysis in modeling tasks using a subset of BPMN, modeling styles were found to depend on both the modeler and the task. Other work has applied process mining to investigate how novice modelers validate their domain models and how this relates to learning outcomes [15, 16]. As LLMs become increasingly integrated into the modeling process, it remains an open question whether these previously identified patterns still hold or whether new ones are emerging — a question explored in part by recent work on modeler–LLM interaction patterns [17]. In what follows, we analyze related work on interaction logging in modeling tools.

Several tools have explored interaction logging beyond traditional web analytics, each at a different point in the design space between raw event capture and actionable behavioral insight (Table 1). On one end sit commercial platforms such as Hotjar [10] and FullStory [9], prioritizing ease of deployment and breadth of capture over configurability or data ownership; both have attracted criticism in the privacy literature, where automated redaction of sensitive form fields was shown to be incomplete despite nominal opt-out signals [18]. On the other end are research instruments such as Interaction Lab [19], which focuses on pointer kinematics that convey information about cognitive effort and hesitation, and WebTrace [20] for remote usability testing. All of these tools operate on raw DOM events and have no notion of modeling operations such as creating a node or establishing a typed connection.

TOOL. Developed at FernUniversität in Hagen since 2013 [4], TOOL is the most closely related prior work: a web-based modeling tool and research observatory built to study how individuals perform conceptual modeling and where they struggle. It ships two graphical editors (an adapted Entity-Relationship variant and a BPMN 2.0 subset) and combines four observation modes that can be switched on or off per study: interaction event tracking with canvas replay, PPM-style dot diagrams and mouse heatmaps; think-aloud verbal protocol recording; WebRTC screen capture; and pre- and post-session surveys [21]. However, it was built for a specific research agenda: the editors support exactly two modeling languages, the observation infrastructure is tightly integrated with those editors rather than exposed as a reusable library, and the monolithic Java EE deployment is not suited for lightweight integration into third-party modeling frameworks.

MER. The Modeling Event Recorder (MER) [5], captures user interactions in EMF-based graphical editors by hooking into the EMF Notification API, supporting both Sirius-based desktop editors and GLSP-based web editors. It serializes sessions into IEEE XES logs, an EMF-based XES model, and an extended EMF Change Model, and has been applied to Ecore modeling and in an industrial case study with Volvo Construction Equipment. MER works at the *model layer*: it captures exactly the operations the editor considers semantically meaningful, but UI-level interactions that do not change the model, like mouse movements, hover dwell, zoom and pan gestures, keyboard shortcuts, failed drag attempts, and gaze fixations, are invisible to it by design.

Research gap. As Table 1 summarizes, only TOOL and MER capture the semantic layer of a graphical modeling editor, and both do so under assumptions, i.e., fixed languages in TOOL and EMF-only model events in MER, that are too restrictive for a general-purpose tracking layer. None of the surveyed tools is designed as a reusable, tool-agnostic library that an arbitrary web-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 here.

Table 1
Comparative overview of existing monitoring tools and frameworks.

Tool (license)	Interaction Capture	Analysis Techniques	Modeling-aware?
PostHog [8] (MIT, ee/ proprietary)	Autocapture of DOM events: clicks, pageviews, form submissions, rage clicks	Session replay, funnels, user paths, retention, heatmaps, A/B tests	No
WebTrace [20] (source avail.)	Clicks, keyboard input, navigation, scrolls; browser extension	Behavior abstraction, per-session inspection, CSV export	No
Hotjar [10] (proprietary)	Clicks, mouse movement, scroll, keystrokes, forms; JS snippet	Click/move/scroll heatmaps, recordings, funnels, frustration signals	No
FullStory [9] (proprietary)	Tagless autocapture of all DOM interactions; web and mobile SDKs	Session replay with retroactive search, funnels, user journeys	No
Interaction Lab [19] (prototype)	Pointer kinematics: speed, acceleration, curvature, corrections	Plugin-based feature extractors, CSV export for offline analysis	No
TOOL [21] (closed)	Semantic canvas events, think-aloud audio, screen capture, surveys	Canvas replay, PPM dot diagrams, mouse heatmaps, qualitative coding	Yes (ER, BPMN)
MER [5] (EPL 2.0)	Model-level events via the EMF Notification API: create, set, delete	XES export, EMF-based XES models, process mining via ProM	Yes (EMF, GLSP)
This work (open source)	Low-level input, semantic GLSP actions, viewport changes, webcam gaze	N-gram clickstream + XES, friction score, gaze heatmaps, replay	Yes (tool-agnostic)

3. Tool Architecture and Implementation

The methodology of this work follows a structured process inspired by Design Science Research (DSR) [22]. The process encompasses the analysis of earlier related work (as reported in Section 2) to identify the current state of the art and research gap, the design and implementation of an artifact to address the gap (Section 3 and 4), and its empirical evaluation in a realistic use case setting (Section 5). This methodology ensures that both theoretical and practical aspects of interaction tracking in web-based modeling tools are addressed, resulting in a validated, reusable framework that supports future UX research and tool development. The literature review yielded a set of functional and non-functional requirements (see Section 3.1) that guided the design of the framework. The framework is described in Section 3.2 and Figure 1. Additionally, the framework provides three analytical pipelines that transform raw interaction data into behavioral information, described in Section 3.3.

3.1. Requirements

Nine functional and five non-functional requirements were derived from the literature survey summarized above and from the comparative analysis of existing monitoring tools and frameworks reported in Table 1. The framework must record both fine-grained input events such as mouse actions and drag-and-drop (FR1) and the higher-level *semantic* actions that reflect user intent (FR2): creation, deletion, modification, and movement of model elements and connections. It should capture view manipulation such as panning and zooming (FR3). It must additionally collect and analyze eye-tracking data (FR4), provide a standardized, extensible event representation carrying timestamps, session context, and metadata (FR5), persist sessions and support their retrieval (FR6), offer analytical methods for interaction patterns and usability issues (FR7), replay recorded sessions (FR8), and export data for external tools (FR9). Non-functionally, it must be modular and extensible (NFR1), independent of any specific modeling tool (NFR2), introduce only limited performance overhead (NFR3), support privacy-conscious collection through consent, data minimization, pseudonymization, access control, and deletion (NFR4), and be compatible with browser-based modeling environments (NFR5).

3.2. Framework Architecture

Backend and Data Model. The backend provides RESTful endpoints for ingestion, persistent storage, and analysis. FastAPI was selected for its asynchronous request handling, validation through Pydantic models, and built-in OpenAPI documentation. Rather than the specialized time-series database initially considered, PostgreSQL is used for all storage: its JSONB support enables flexible storage of heterogeneous event payloads, its indexing gives adequate query performance at the expected data volumes, and its operational complexity is far lower.

The schema is centered on three tables. The `recording_sessions` table is the header of a recording: which tool produced the data, what model or editor was open, who recorded it, when the session started and ended, plus summary values and an `extra_data` field for tool-specific metadata. It is intentionally generic, using field names such as `tool_id` and `editor_type` rather than tool-specific terminology, allowing the same backend to serve multiple GLSP-based tools without modification. The `interaction_events` table stores the interaction stream, ordered by timestamp and sequence number, recording a coarse category (`event_type`) and optional subtype (`event_kind`) while the complete payload is preserved in a JSONB `data` column; frequently queried fields such as element identifiers and coordinates are duplicated into dedicated columns for efficient indexing. The `gaze_points` table stores raw eye-tracking samples separately and is optimized for inserts and time-based queries.

Session upload is atomic: the primary endpoint accepts a complete session as a single JSON payload rather than streaming individual events, which simplifies the client and guarantees consistency. The backend denormalizes key information from the payloads on ingestion. All write endpoints require an API key, and deletion endpoints support GDPR compliance through cascading deletes.

Frontend Tracking Library. The client-side library is implemented in TypeScript and separates event capture, session management, and data export. The central `InteractionTracker` service maintains the captured event array, the session metadata, and disposable resources for cleanup; when a session starts, it detects the diagram type from the active file, extracts the model file path, and captures workspace context. The library defines an enumeration of interaction event types spanning diagram interactions (selection, creation, deletion, movement, resizing, editing), view changes, property modifications, file operations, and session lifecycle events.

Its most critical capability is intercepting GLSP protocol actions. Rather than relying solely on DOM events, the library hooks into the GLSP action dispatch mechanism for both client-originated actions and server responses, so that user-initiated operations and server-confirmed model changes are both captured, and translates them into standardized events: node creation captures the element type identifier, container reference, and location; edge creation records source and target identifiers; selection tracks selected and deselected collections; label edits capture the label identifier and new text.

Viewport changes occur at high frequency and would otherwise dominate the event stream, so the library debounces them (default 100 ms), reducing hundreds of events per manipulation to a single event representing the final state. A dedicated `ViewportTrackingHandler` at the GLSP client level captures scroll position and zoom level, since this is not accessible from the extension layer, and an `ElementBoundsTrackingHandler` closes a gap found in initial testing, where bounds changes of elements moved inside their parent containers were not captured. On session end, the data is exported as JSON (preserving the nested structure for replay) and as CSV.

Eye-Tracking Integration. Early development attempted to embed the webcam-based eye tracker (WebGazer.js) directly into the GLSP client running inside the VS Code webview. This ran into three obstacles: the restrictive Content Security Policy of webviews complicated camera access and the loading of WebGazer's TensorFlow models; real-time gaze prediction in the same process as the diagram renderer caused noticeable lag; and WebGazer calibration requires clicking reference points across the whole screen, whereas the canvas occupies only part of the window.

Eye-tracking was therefore implemented as a standalone browser application in a separate window, which provides full-screen calibration coverage, isolation of gaze processing, unrestricted access to

the browser API, and the flexibility to switch eye-tracking technologies without modifying the main application. It is a single HTML file with a status display, controls for tracking, calibration, and export, and a 9-point calibration overlay that enforces sequential clicking and records five samples per point; WebGazer uses ridge regression for gaze prediction and the TFFacemesh tracker for face detection. Gaze data is exported as JSON and joined with the interaction stream in the backend through the session identifier.

Interaction Replay. The framework can replay a recorded session, reconstructing the modeling process from the captured events. It consists of an event parser that reads the session log and filters negligible events, an action synthesizer that translates logged events back into GLSP protocol actions, and an ID mapping engine. The latter addresses the main technical challenge: GLSP assigns UUIDs at creation time, so the same conceptual element carries different identifiers in the original and the replay session. A three-phase algorithm resolves this. In *pre-analysis*, the event sequence is scanned before replay to associate creation operations with element IDs, exploiting the temporal ordering. IDs first appear in selection or property events shortly after creation. During *runtime ID capture*, the system listens for the server's `updateModel` response after each creation and captures the newly assigned ID. In *ID translation*, subsequent actions have their identifiers translated before dispatch. Container hierarchy is handled by tracking the most recently selected element of each container type.

3.3. Analytical Pipelines

Three pipelines transform raw interaction data into behavioral information.

Clickstream analysis and XES export. Clickstream analysis treats user interactions as an ordered sequence of discrete events [23]. Given a sequence $E = (e_1, \ldots, e_m)$, the analyzer computes n-grams for all contiguous subsequences: $\text{n-grams}(E, n) = \{(e_i, \ldots, e_{i+n-1}) \mid 1 \leq i \leq m - n + 1\}$; bigram frequencies expose direct transitions, trigrams capture slightly longer patterns, and a transition matrix T records how often event e_j immediately followed e_i. Since event granularity strongly influences the results, the analyzer accepts a granularity parameter producing labels of the form `event_type`, `event_type:event_kind`, or `event_type:event_kind:element_type` (e.g. `element_create:createNode:Class`). Before computing n-grams, `mouse_click` and `element_select` events are filtered out: they are frequent enough to dominate pattern statistics without reflecting intentional modeling decisions. Sessions are exported to the IEEE 1849-2016 XES format [24] with the Concept, Time, and Lifecycle extensions registered, sessions mapped to traces, and event names at full granularity; the files are compatible with ProM, Disco, and PM4Py [25].

Friction detection. User friction refers to any interaction pattern that indicates difficulty, confusion, or frustration during system use; behavioral metrics often reveal usability issues that users themselves fail to articulate [26]. Four indicators are operationalized. *Rage clicking*, i.e., repeated rapid clicks on an element, typically three or more within a short window [27], is detected with a sliding window (default 1500 ms) validated against a maximum radius from the cluster centroid (default 50 px). *Undo/redo bursts* indicate that users struggle to achieve the intended outcome and frequently precede task abandonment [28]; a pattern string (`U` = undo, `R` = redo) is built, and severity is boosted for oscillating patterns such as URURUR. *Navigation thrashing*, i.e., erratic viewport changes that emerge when users cannot locate a target or maintain orientation, is flagged when multiple viewport events occur within the configured window (default 5 s), and *rapid element deletion* clusters (default 3 s) signal that users are abandoning their current approach. Following the practice of combining usability metrics into unified scores [29], the four component scores are weighted (rage 0.35, undo/redo 0.30, navigation 0.20, deletion 0.15) into a composite score from 0 to 100, mapping to the categories minimal (0–20), low (21–40), moderate (41–60), high (61–80), and critical (81–100) friction. All thresholds and weights are configurable.

Table 2
Canonical `event_type` values and the pipelines that consume them (C = clickstream, X = XES export, F = friction, M = metadata only, — = filtered out as noise).

type	Semantics		type	Semantics	
`element_select`	Selection of elements.	—	`property_change`	Property-palette edit.	C,X
`element_create`	Node or edge creation.	C,X	`viewport_change`	Pan/zoom, debounced.	C,F
`element_delete`	Element deletion.	C,X,F	`zoom_change`	Zoom, if split from pan.	C,F
`element_move`	Bounds change, position.	C,X	`scroll`	Scroll within the canvas.	F
`element_resize`	Bounds change, size only.	C,X	`text_edit`	Free text, not a label.	C,X
`element_edit`	Compound or label edit.	C,X	`mouse_click`	Raw pointer click.	F
			`session_start/end`	Session bounds.	M

Gaze heatmaps. Raw gaze recordings are too detailed to interpret directly, so heatmaps aggregate gaze points into a spatial overview of the distribution of attention. The pipeline constructs a two-dimensional histogram of gaze coordinates and convolves it with a Gaussian kernel that balances visual coherence against spatial precision; following recommendations for typical screen-based setups [30] the default spatial precision is 18 px, configurable from 1 to 100 through the API. Values are normalized to $[0, 1]$ and rendered with the sequential *hot* colormap rather than a perceptually misleading rainbow scale [31], and, where available, the heatmap is overlaid semi-transparently (default $\alpha = 0.45$). Heatmaps are generated on demand and streamed as PNG rather than pre-computed, which keeps storage minimal and allows regeneration with different parameters.

4. Reference Integration and Third-Party Adoption

BIGUML integration. The framework was integrated into BIGUML [11], a GLSP-based UML modeling tool built according to the reference architecture of Metin and Bork [3]. It supports nine UML diagram types and runs as a VS Code extension with a webview-hosted GLSP client and a Java-based GLSP server. The tracking library was added as a new package in BIGUML's monorepo, depending on GLSP protocol and client libraries at version 2.2.0. The `InteractionTracker` and `InteractionReplayService` are bound as singletons in the InversifyJS container; a dedicated container module registers the viewport and bounds handlers as `IActionHandler` implementations in the GLSP client. A sidebar panel provides buttons to start and stop tracking, launch the standalone eye-tracking application, and upload sessions.

Three lessons emerged for future GLSP integrations. *Action protocol variability*: GLSP actions use inconsistent property names — `elementTypeId` for creation, `sourceElementId`/`targetElementId` for edges, `elementIds` for deletion, `elementId` for updates, `labelId` for label edits — requiring multiple code paths, and compound actions need recursive processing. *Container ID management*: the original implementation recorded the diagram root's UUID, which differs between recording and replay; a `$ROOT` placeholder now signals the replay system to omit it. *Type system*: BIGUML uses identifiers such as `CLASS__Class` while other tools differ, so the library implements flexible type matching.

Integrating another frontend. The backend is deliberately tool-agnostic at the wire level. A third-party frontend needs only `POST /sessions` to upload a complete session envelope, optionally `POST /sessions/{id}/gaze` for the gaze stream, and `DELETE /sessions/{id}` for GDPR-compliant removal. A small set of fixed fields is parsed, indexed, and consumed by the analytical pipelines; everything else is preserved unchanged in the JSONB columns, which is where tool-specific concepts live. The pipelines do not care which GUI gesture produced an event, only about its `type`, optionally its `kind`, and its element type. Table 2 lists the canonical `event_type` values and how each pipeline treats them; reusing them — and the exact `kind` strings the friction detector recognizes (`undo`, `redo`, `setViewport`, `center`, `fit`, `scroll`, `zoom`, `deleteElement`) — is sufficient for the existing detection logic to fire without modification. New types may be added freely, but are ignored by the friction detector and excluded from the XES export until the pipelines are updated.

5. Exploratory Evaluation

5.1. Setup

To evaluate the framework's practical applicability, a small exploratory user study was conducted. Its goal was not to obtain statistically significant results, but to demonstrate that the framework can capture, reconstruct, and analyze user interactions in a realistic scenario. Seven participants, all with a technical background and basic familiarity with diagram-based modeling but varying levels of experience, completed the same task using the instrumented BIGUML editor; six sessions had eye-tracking enabled.

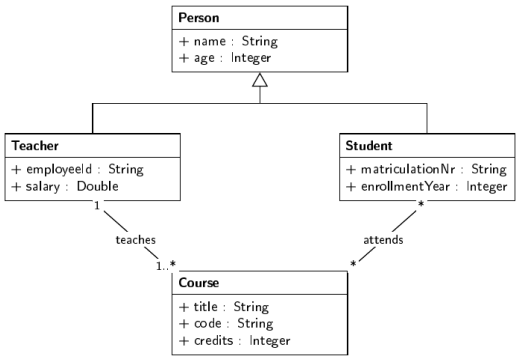


Figure 2: Reference UML class diagram used in the study.

The task consisted of recreating the class diagram shown in Figure 2: four classes (*Person*, *Teacher*, *Student*, *Course*), an inheritance hierarchy in which *Teacher* and *Student* specialize *Person*, and two labeled associations. Each class defined a small set of attributes so that participants would exercise the most common interaction patterns of a class diagram editor; more advanced features were omitted to keep the focus on interaction behavior rather than on modeling complexity. Participants received a short introduction to basic tool operation, but the reference diagram was not demonstrated, and no modeling sequence was recommended, to minimize bias towards a particular approach, and no time limit was imposed. The raw JSON exports of all seven sessions, including the six gaze streams, are published on Zenodo without post-processing, filtering, or aggregation [12].

5.2. Session Overview

Table 3 summarizes the headline metrics; all participants completed the task and produced a diagram closely matching the reference. Across the dataset the backend stored 1 440 interaction events spanning 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. Two observations follow. 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 — 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 re-attached through the session identifier, confirming that the tracking lifecycle and the eye-tracking integration behave robustly.

5.3. Sequential Modeling Patterns

Applying the n-gram analysis at granularity `event_type:event_kind:element_type` reveals two qualitatively distinct modeling strategies across the seven participants, classified by the relative weight of property-related events (`createNode:Property`, `applyLabelEdit`, `updateElementProperty`) versus layout-related events (`changeBounds`, `changeRoutingPoints`, `element_edit:compound`) in the filtered event stream.

The *property-centric* strategy (P2, P5, P6) is dominated by the transitions `createNode:Property` → `applyLabelEdit` (or `updateElementProperty`, depending on whether the label was edited 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 is the strongest version of the pattern, containing *not a single* layout-related event in 145 events. The shared workflow is: create class, add and label all of its attributes, move on to the next; connections are drawn only after most classes are populated.

Table 3
Per-session summary: duration, captured events, eye-tracking (ET), dominant modeling strategy, composite friction score (0–100), detected rage-click incidents (RC ev.), and the total 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

The *layout-centric* strategy (P1, P3, P4, P7) shows a completely different signature. The most frequent bigram of distinct user operations for P1, P3, and P4 is `element_move:changeBounds` → `element_edit:compound` (12, 12, and 6 occurrences), and the most frequent trigrams are the alternation `changeBounds` → `compound` → `changeBounds`: these participants alternate between repositioning a class and committing the change, preferring to fix the spatial layout before completing the semantic content. P4 and P7 add heavy use of `changeRoutingPoints` (6 and 9 occurrences), manually adjusting the path of generalization and association edges. This reflects behavior the property-centric participants did not do at all.

The classification correlates loosely with duration: the four layout-centric participants completed the task in a mean of 254 s, the three property-centric ones in 210 s. A 21 % slowdown is plausible given the additional repositioning and routing work, but the variance within each group (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. Methodologically, the chosen granularity is fine enough to separate property-driven from layout-driven workflows, yet coarse enough to keep the dominant n-grams stable and readable across participants.

5.4. Friction Detection

All sessions produced a composite friction score in the lower band (six at 35.0, P3 at 39.5; mean 35.6; category *low friction*), but the substructure of these scores is the more informative finding. 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, so the composite scores are driven almost exclusively by rage-click events: 550 rapid clicks across 114 detected incidents. The clicks consistently target the node bodies of recently created `CLASS__Class` or `CLASS__Property` elements, with a typical timing pattern of 4–6 clicks within 200–1500 ms. We hypothesize that this is caused by attempts to rename or re-select a freshly created element while the inline edit affordance was not yet active or not where they expected it.

This consistency across all seven sessions, regardless of strategy, strongly suggests that thresholds 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; a higher minimum click count or a tool-specific exclusion of in-place edit affordances would be a sensible adjustment, which the configurable thresholds permit. P7 is the only session in which clusters reached *critical* severity: 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, both in the upper canvas within a few seconds of each other — exactly the kind of localized, time-bounded usability incident the framework is intended to surface. Aggregating the reported spatial centroids yields four hotspot clusters that align with the layout of the reference diagram, showing that friction concentrates on the elements participants were

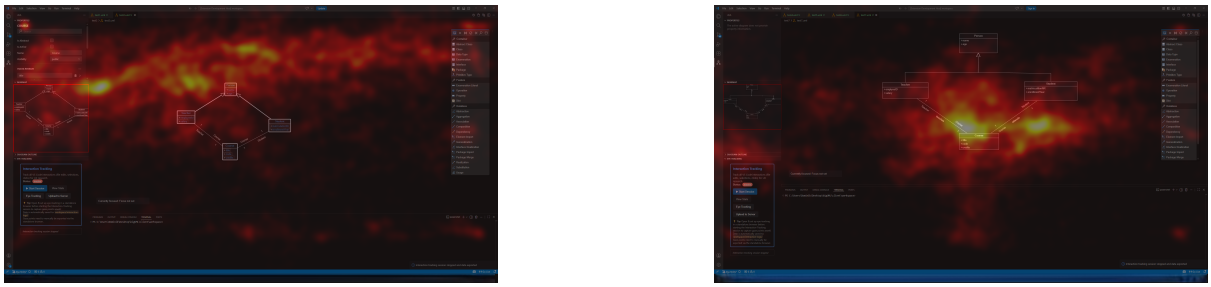


Figure 3: Aggregate gaze heatmaps overlaid on a screenshot of the BIGUML editor. Left: P2 (property-centric, 351 s). Right: P7 (layout-centric, 291 s, the event-richest session), whose strongest concentrations follow the inheritance hierarchy and the path of the *Course* associations. Warmer colors indicate higher gaze density.

actively creating and editing. Beyond rage clicks, P3 produced a single navigation-thrashing incident (six viewport changes within 2.2 s, total travel ≈ 1452 px) shortly before drawing the last generalization edge, matching its layout-centric classification; the absence of undo/redo activity is consistent with the low task complexity, but also means this indicator could not be exercised here.

5.5. Visual Attention

For the six eye-tracking sessions, the standalone WebGazer application produced gaze streams rendered as semi-transparent attention heatmaps overlaid on the editor screenshot. Figure 3 shows two of these, chosen to span the two modeling strategies. Three patterns are visible. First, attention is heavily concentrated on the central canvas region where the diagram is being constructed. 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 regions were used in a *glance-and-return* manner. Third, the layout-centric participants spread their attention across the inter-class regions — labels, routing points, association names — substantially more than the property-centric P2, whose hot zone concentrates around the class bodies. This leads to the hypothesis that the strategy distinction visible at the event-stream level also has a counterpart at the visual-attention level, and 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 — a hypothesis a larger, controlled study could confirm or refute. Two limitations apply throughout: WebGazer accuracy of approximately 100–150 px resolves regions but not individual controls, and temporal dynamics are lost in the aggregate view.

5.6. Process-Mining Interoperability

The recorded data allows the use of process mining techniques for modeling actions. The data can also be exported as XES, which enables out-of-the-box use of process-mining tools such as ProM and Disco without additional transformation—the XES log is simply imported into the tools. In ProM, each session appears as a trace of chronologically ordered semantic modeling events, with timestamps and metadata available for further analysis. Inspecting P2's trace revealed the creation of a superfluous abstract class, followed by its deletion. Figure 4 shows P7's session as a process model discovered by Disco, where nodes represent recorded semantic activities and edges represent observed transitions, with more prominent nodes and connections indicating higher frequency. The transition from `createNode:Property` to `property_change:applyLabelEdit`, for instance, shows that newly created properties were consistently renamed right after creation. Beyond logging, the framework is designed to support direct integration of process mining into modeling tools — enabling future work on agentic MDE and smart modeling assistance that leverages process mining metrics and directly-follows graphs.

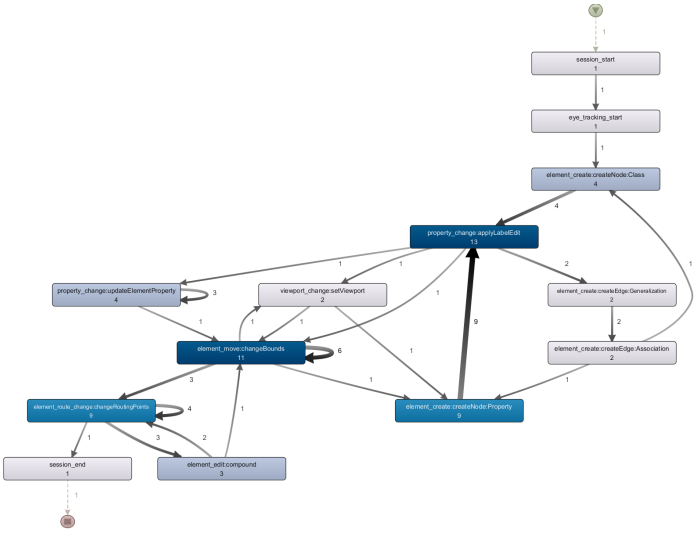


Figure 4: Process model generated by Disco from participant P7's exported XES session. see the online material[2] for a high-resolution version of the figure.

6. Discussion

Requirements assessment. The study provides direct evidence for all nine functional requirements: Complete interaction streams cover FR1–FR3, the six gaze-enabled sessions cover FR4, recorded actions were transformed into the common event schema while the JSONB payload preserved tool-specific information (FR5), sessions were uploaded, persisted, and retrieved through the REST API (FR6), the three pipelines produced interpretable findings (FR7), the replay component reconstructed sessions and validated tracking completeness (FR8), and XES logs were imported into ProM and Disco (FR9). Tool agnosticism (NFR2) is reflected in a backend that does not depend on BIGUML structures and in a generic integration contract, but BIGUML remains the only tool fully integrated; mechanisms for reducing runtime overhead exist (NFR3), and no blocking performance issue arose, but no controlled latency measurement was conducted; and privacy (NFR4) is addressed through opt-in tracking, local-by-default storage, transparent exports, authenticated access, and session deletion, but no full data-protection impact assessment was performed.

Limitations. Seven sessions suffice to demonstrate that the framework captures, structures, and analyzes interaction data as intended and to surface concrete tool-design hypotheses, but not to support quantitative claims about the prevalence of either modeling strategy or the absolute level of friction in BIGUML relative to other tools. The friction thresholds were inherited from general-purpose web applications, and the study itself showed they are too lenient for diagram editors. Finally, the eye-tracking pipeline is not yet end-to-end automated: gaze data must be exported manually, and screen coordinates are not yet transformed into canvas coordinates accounting for pan and zoom.

7. Conclusion and Future Work

This paper presented an open-source, modular framework for tracking and analyzing user interactions in web-based modeling tools. It 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, and that the resulting event streams can be turned into three complementary forms of behavioral information. Even at small scale, the framework surfaces non-trivial patterns: two qualitatively different modeling strategies emerged from an identical task, and the friction analysis potentially flagged a saturated rage-click signal pointing to a concrete inline-edit affordance issue rather than general user frustration — neither visible from session timing alone.

These results allow researchers and modeling tool developers to analyze modeler behavior, contributing to advances in LLM-assisted conceptual modeling. For example, insights from modeler behavior analysis can lead to personalized recommendations based on a user's modeling habits. This work is developed in the context of the Smart GLSP project, which aims to advance LLM-assisted support in GLSP-based modeling tools[6].

Four directions for future work appear promising. The most immediate is a larger study, enabling a more robust comparison of modeling strategies and a baseline for a focused friction-threshold recalibration experiment. A second is closing the eye-tracking pipeline: automatic upload of gaze streams, screen-to-canvas coordinate transformation, and fixation detection at ingestion time, which would move the analysis from raw gaze density to fixation-based metrics comparable with the eye-tracking literature. A third is higher-order analytics, e.g., conformance checking against a reference modeling workflow, time-to-completion analysis per element, and predictive modeling of task abandonment from early-session features, which would turn the framework from an observatory into a diagnostic instrument; the kinematic features pioneered by Interaction Lab [19] are an obvious complement on the input side. A fourth is ecosystem validation: integrating the library into further GLSP-based or non-GLSP-based editors would substantiate the tool-agnostic claim and enable comparative evaluation.

Declaration on Generative AI

During the preparation of this work, the authors used Claude (Anthropic) to generate initial boilerplate code for the FastAPI backend (endpoint structures, Pydantic schemas, SQLAlchemy model templates) and to analyze console log output during debugging, and the Overleaf AI Assistant to paraphrase individual paragraphs. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content. The conceptual design, architectural decisions, implementation logic, and argumentation remain the work of the authors.

Acknowledgments

This study was partially funded by the FFG-funded project SmartGLSP – Facilitating Large Language Models for Smart GLSP-based Modeling (Grant number: FO999925707).

References

[1] D. Bork, P. Langer, T. Ortmayr, A vision for flexible glsp-based web modeling tools, in: The Practice of Enterprise Modeling: 16th IFIP Working Conference, PoEM 2023, Proceedings, volume 497, Springer, Cham, 2023, pp. 109–124. doi:10.1007/978-3-031-48583-1_7.

[2] R. Kalantari, T. C. Lethbridge, Characterizing UX Evaluation in Software Modeling Tools: A Literature Review, IEEE Access 10 (2022) 131509–131527. doi:10.1109/ACCESS.2022.3227504.

[3] H. Metin, D. Bork, A reference architecture for the development of GLSP-based web modeling tools, Softw. Syst. Model. 24 (2025) 1869–1895. doi:10.1007/S10270-024-01257-Y.

[4] B. Ternes, S. Strecker, A web-based modeling tool for studying the learning of conceptual modeling, Gesellschaft für Informatik e.V., 2018, pp. 325–328. URL: https://dl.gi.de/handle/20.500.12116/14954.

[5] M. Dehghani, L. Berardinelli, M. Wimmer, Towards Modeling Process Mining for Graphical Editors, in: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, IEEE, 2023, pp. 929–933. doi:10.1109/MODELS-C59198.2023.00146.

[6] A. Bansah, D. Agyei, Perceived convenience, usefulness, effectiveness and user acceptance of information technology: evaluating students' experiences of a Learning Management System, Technology, Pedagogy and Education 31 (2022) 431–449. doi:10.1080/1475939X.2022.2027267.

[7] J. R. Bergstrom, A. Schall, Eye Tracking in User Experience Design, 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2014.

[6] https://smartglsp.vercel.app/

[8] PostHog, PostHog – We make dev tools for product engineers, 2026. URL: https://posthog.com/.
[9] Fullstory, FullStory, 2026. URL: https://www.fullstory.com/platform/session-replay/.
[10] Hotjar, Hotjar: Website Heatmaps & Behavior Analytics Tools, 2026. URL: https://www.hotjar.com/.
[11] H. Metin, D. Bork, Introducing bigUML: A flexible open-source glsp-based web modeling tool for uml, in: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2023, pp. 40–44. doi:10.1109/MODELS-C59198.2023.00016.
[12] D. Granig, Interaction tracking dataset for web-based modeling tools, https://doi.org/10.5281/zenodo.20112314, 2026.
[13] J. Pinggera, P. Soffer, D. Fahland, M. Weidlich, S. Zugal, B. Weber, H. A. Reijers, J. Mendling, Styles in business process modeling: an exploration and a model, SoSyM 14 (2015) 1055–1080.
[14] J. Pinggera, M. Furtner, M. Martini, P. Sachse, K. Reiter, S. Zugal, B. Weber, Investigating the process of process modeling with eye movement analysis, in: M. La Rosa, P. Soffer (Eds.), Business Process Management Workshops, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 438–450.
[15] G. Sedrakyan, J. De Weerdt, M. Snoeck, Process-mining enabled feedback: "tell me what i did wrong" vs. "tell me how to do it right", Computers in Human Behavior 57 (2016) 352–376.
[16] G. Deeva, M. Snoeck, J. De Weerdt, Discovering the impact of students' modeling behavior on their final performance, in: R. A. Buchmann, D. Karagiannis, M. Kirikova (Eds.), The Practice of Enterprise Modeling, Springer International Publishing, Cham, 2018, pp. 335–350.
[17] S. J. Ali, I. Reinhartz-Berger, D. Bork, How are LLMs used for conceptual modeling? An exploratory study on interaction behavior and user perception, in: Conceptual Modeling - 43rd International Conference, ER 2024, Pittsburgh, PA, USA, October 28-31, 2024, Proceedings, Springer, 2024, pp. 257–275. doi:10.1007/978-3-031-75872-0_14.
[18] G. Acar, S. Englehardt, A. Narayanan, No boundaries: data exfiltration by third parties embedded on web pages, Proceedings on Privacy Enhancing Technologies 2020 (2020) 220–238.
[19] D. Fernández-Lanvin, J. De Andrés, M. González-Rodríguez, P. Torre, Interaction Lab: Web User Interaction Tracking and Analysis Tool:, in: Proceedings of the 18th International Conference on Web Information Systems and Technologies, 2022, pp. 152–158. doi:10.5220/0011567100003318.
[20] J. Pyun, M. Heo, Y. Choe, J. Jeong, WebTrace: An enhanced system for remote user interaction tracking and analysis in web applications, SoftwareX 30 (2025) 102075. doi:10.1016/j.softx.2025.102075.
[21] K. Rosenthal, B. Ternes, S. Strecker, Understanding individual processes of conceptual modeling: A multi-modal observation and data generation approach, in: Modellierung 2020, Gesellschaft für Informatik e.V., 2020, pp. 77–92. URL: https://dl.gi.de/handle/20.500.12116/31848.
[22] A. Hevner, S. March, P. Jinsoo, S. Ram, Design Science in Information Systems Research, Management Information Systems Quarterly 28 (2004) 75.
[23] W. Aalst, Process Mining: Data Science in Action, 3 ed., Springer, 2016. doi:10.1007/978-3-662-49851-4.
[24] IEEE, 1849-2016 - IEEE Standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams, 2016.
[25] A. Berti, S. van Zelst, W. Aalst, Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science (2019). doi:10.48550/arXiv.1905.06169.
[26] T. Tullis, B. Albert, Measuring the User Experience (Second Edition), Interactive Technologies, Morgan Kaufmann, 2013.
[27] Fullstory, What are Rage Clicks? How to Identify Frustrated Users | Fullstory, 2022. URL: https://www.fullstory.com/blog/rage-clicks/.
[28] D. Akers, M. Simpson, R. Jeffries, T. Winograd, Undo and erase events as indicators of usability problems, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09, Association for Computing Machinery, 2009, pp. 659–668. doi:10.1145/1518701.1518804.
[29] J. Lewis, J. Sauro, Quantifying the User Experience, 2 ed., 2016.
[30] A. Bojko, Eye Tracking the User Experience: A Practical Guide to Research, 2013.
[31] D. Borland, R. M. Taylor Ii, Rainbow Color Map (Still) Considered Harmful, IEEE Comput. Graph. Appl. 27 (2007) 14–17. doi:10.1109/MCG.2007.323435.