Beyond Interface Generation: GenUI as Interpretive Runtime

MATTIAS ROST, University of Gothenburg, Sweden

I argue that generative UI moves beyond design-time or use-time UI generation: from application-centered systems with pre-specified representational boundaries to an *interpretive runtime* in which interface and capability are continuously re-specified through interaction. The UI is not merely rendered from a predefined design space but emerges as a provisional stabilization of evolving intent, contextual reasoning, and executable functions. I outline three properties—provisionality, intent negotiation, and executable emergence—and discuss implications for HCI practice.

Additional Key Words and Phrases: Generative UI; LLM-mediated computing; interpretive runtime

1 Introduction

Generative UI (GenUI) is typically discussed along two trajectories: AI-augmented workflows that assist designers during production, and AI systems that generate interfaces at run-time [1]. Both matter. However, I argue that GenUI offers a deeper transformation: a shift from application-centered computing—where interfaces are largely stabilized through design-time specification—to a computational condition in which interfaces are composed and revised at run-time in response to evolving intent and contextual reasoning. I call this condition an *interpretive runtime*. In the broader frame of *LLM-mediated computing* [3], GenUI is a visible manifestation of a shift toward *point-of-use* interface production, where AI systems increasingly deliver UI directly to end-users and computation becomes organized around ongoing interpretation and re-specification.

2 Interpretive Runtime

Work on adaptive and *malleable* software has long emphasized use-time change: interfaces can be configured, personalized, and recombined to better fit situated practice [2]. In many systems, malleability is operationalized through pre-specified constraints: predefined primitives, data models, and transformation rules that determine what counts as a valid change. GenUI shifts malleability from use-time adjustment *within* a predefined design space to run-time composition of interaction structures at point of use, where the interface can change during the interaction and control over change may be user-driven, system-driven, or negotiated.

Interpretive runtime names a computational condition in which the interface is not the presentation layer of a pre-specified application but a provisional stabilization of an ongoing interpretive process. In *LLM-mediated computing*, a system can translate user expressions and actions into executable procedures (e.g., scripts, workflows, tool calls) and render interfaces that are tightly coupled to those procedures. Crucially, this coupling is revisable: as users clarify intent, introduce constraints, or encounter breakdowns, the system may re-specify both the underlying procedure and the interface that makes it actionable.

3 Three Properties of Interpretive Runtime

There are three key properties of an interpretive runtime:

Provisionality: In an interpretive runtime, interfaces are temporary stabilizations rather than final artifacts. A UI is "good enough for now", until intent, constraints, or context shifts and a different stabilization becomes preferable.

Author's Contact Information: Mattias Rost, University of Gothenburg, Gothenburg, Sweden, mattias.rost@gu.se.

Intent Negotiation: User intent is not treated as fully given or specified. Instead, it is interactionally negotiated through clarification, breakdown, and repair. The system's outputs are part of how intent is articulated, not merely how it is executed.

Executable Emergence: An interpretive runtime is not only about rendering widgets, it is about making emergent computational structures actionable. The system assembles procedures and representations together, so that new capabilities appear through interaction.

4 Implications for GenUI Practice

GenUI can be considered the visible layer of an interpretive runtime. Designing for such runtimes shifts attention from specifying UI to specifying *conditions* for safe, legible, and repairable emergence.

- **Design invariants:** specify constraints, permissions, data contracts, and safety boundaries that hold across generated interfaces.
- **Make generation inspectable:** provide traces of what was generated, why, and what resources or tools were invoked, to support accountability.
- **Support repair and rollback:** enable undo, branching histories, and reversible execution so users can recover from misinterpretations.
- **Evaluate interpretive coupling:** assess misalignment, drift, recoverability, and the quality of clarification, not only interaction efficiency.

5 Research Questions

In thinking about interpretive runtime and LLM-mediated computing, I propose the following research questions relevant for the workshop:

- What counts as *consistency* when the UI can change during interaction and across sessions?
- What enables *skilled coping* and confidence-in-action when the system's surface and operations keep shifting?
- What *representations* (histories, traces, contracts) make interpretive runtime accountable and legible?
- How should we design to prevent fluent generation from becoming persuasive improvisation (e.g., dark patterns via confidence)?

6 Conclusion

GenUI can be understood as more than interface generation: it is a surface manifestation of interpretive runtime, where computation is organized around run-time interpretation and re-specification rather than execution within fixed representational boundaries. This framing invites HCI to treat GenUI as a paradigm of run-time mediation and to redirect design attention toward invariants, inspectability, repair, and evaluation of interpretive coupling. I offer interpretive runtime as a concept for workshop discussion and as a lens for articulating what is new in GenUI practice.

References

[1] Siân Lindley, Jack Williams, Yining Cao, Haijun Xia, Elizabeth F. Churchill, Abigail Sellen, Jeffrey Nichols, and David R. Karger. 2026. What does Generative UI mean for HCI Practice?. In *Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems* (Barcelona, Spain) *(CHI EA '26)*. ACM, New York, NY, USA, 1–7. doi:10.1145/3772363.3778757

[2] Geoffrey Litt, Josh Horowitz, Peter van Hardenberg, and Todd Matthews. 2025. *Malleable Software: Restoring User Agency in a World of Locked-Down Apps*. https://www.inkandswitch.com/essay/malleable-software/ Accessed: 2026-02-12.

[3] Mattias Rost. 2025. Reclaiming the Computer through LLM-Mediated Computing. *ACM Interactions* 32, 5 (Aug. 2025), 26–31. doi:10.1145/3747585