

Structural Interaction for Generative User Interfaces

Vincent Cavez
Stanford University
USA
vcavez@stanford.edu

Abstract

Generative user interfaces (genUI) promise to produce and adapt interface structures on-the-fly, yet current approaches lack a principled vocabulary for specifying how generated structures should behave during interaction. We argue that the central design challenge for genUI is not what structure to generate but how each structural rule should respond to user action. We present Structural Interaction, a framework that defines structure as the set of rules in a directed graph of elements and rules, and characterizes rule behavior along two orthogonal dimensions: rigidity (how much the user can shape the rules) and enforcement (how much the user is bound by the rules during interaction). Four values per dimension generate a 16-cell design space that enables designers and generative systems to reason precisely about when structure should constrain, adapt, or yield. We discuss how this framework reframes genUI design and outline the shifts HCI practice needs to support it.

Keywords

structure, flexibility, malleable interfaces, interaction design, theoretical framework

ACM Reference Format:

Vincent Cavez. 2026. Structural Interaction for Generative User Interfaces. In . ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/nnnnnnn>. nnnnnnn

1 Motivation and Background

Productivity and creativity support tools rely on rules that govern how users create, organize, and transform content [9]. We call these rules structure. A spreadsheet's grid, a score's staff system, a design tool's layer hierarchy are all structures. Structure simultaneously enables and constrains: the grid enables formulas and sorts precisely because it enforces a rigid schema, but that rigidity obstructs users when their intentions diverge from its logic [5].

As generative systems become capable of producing and adapting interface structures on behalf of users [4], the critical question shifts: how should generated structure behave during interaction? Several frameworks address this organizational layer. Assemblies in DIRA [2] describe what organizes representations and connects interaction techniques, but without specifying behavioral properties.

Interaction substrates [7] define structured environments where users can adjust individual constraints through tweaking and create reusable configurations through templating, but the question of how a constraint responds when users push against it during interaction remains implicit. Malleable software [6, 8] demonstrates that users can reshape interface structure at runtime, but treats malleability as a platform-level property rather than a per-rule design parameter.

In the genUI space specifically, Cao et al. [4] propose generative interfaces driven by task-driven data models that an LLM generates and evolves, demonstrating that structure can drive interface generation. Cao et al. [3] design co-creation environments around compositional structures, addressing which structures to deploy and how to connect them. Their user evaluation confirms the value of making structure a primary design concern, but also surfaces problems that structural composition alone does not resolve: participants sometimes struggled to revise, iterate, and explore freely within the structures provided. These problems arise not from poor structure selection but from the absence of a vocabulary for specifying how generated rules should respond during interaction. There is no shared way to say “this rule should yield when the user pushes against it” or “this rule should be adjustable but not removable.”

2 Structural Interaction

The Structural Interaction framework [10] models the user interface as a directed graph containing two types of nodes: elements and rules. Elements are the entities users perceive: a spreadsheet cell, a note on a musical staff, a layer in a compositing tool. Elements are always targets in the graph, never sources: they do not trigger or constrain anything. Rules are the organizational nodes of the graph, and structure is defined as the set of all rules governing the organization of elements.

Rules are further divided into two types. Declarative rules are constraints that hold continuously and need no trigger: “cell B1 always displays the value of A1 multiplied by two,” “all elements in a layer inherit that layer's opacity,” “this panel's width never falls below 200 pixels.” Imperative rules are conditions that, if triggered, can perform an action: “if the user clicks this button, change its color,” “if a row is dragged past another, swap their positions.” This graph model distinguishes structure from content, presentation, and layout. Because elements are always targets, every transformation in the interface ultimately acts on or through rules: users never interact with elements directly; they interact with structure.

Within any structure, closely related rules cluster into sub-structures that users experience as a unit. A spreadsheet's grid logic is a sub-structure: a cluster of declarative and imperative rules governing how cells are created, addressed, referenced, and manipulated. Every rule carries a behavioral state defined by two orthogonal dimensions.

Permission to make digital or hard copies of all or part of this work for personal or internal use, or for the internal or personal use of specific clients, is granted by ACM for users registered with ACM, provided that the fee of \$12.00 is paid directly to ACM. This permission is granted without fee where organizations have been granted a special rate code by ACM. For all other use, permission should be sought from ACM. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, Washington, DC, USA

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
<https://doi.org/10.1145/nnnnnnn>

Rigidity captures how much the user can shape the rules. Four values span a spectrum from system control to user control. A **fixed** rule is immutable: the email schema in Outlook requires a sender, a recipient, a subject, and a body, and no other rule can modify this. A **negotiable** rule can be relaxed within bounds but not redefined: the volume limiter on iOS lets the user adjust the maximum level within a fixed range, but not redefine how limiting works. A **malleable** rule can be redefined entirely by the user, like a keyboard shortcut in Photoshop that can be reassigned to any function. An **authorable** rule goes further: the user can create rules that did not previously exist, as when a user writes a macro in Excel or defines custom geometric constraints in Fusion 360.

Enforcement captures how much the user is bound by the rules during interaction. Four values span a spectrum from unyielding to yielding. A **persistent** rule does not yield: a minimum panel width blocks hard at its limit. An **elastic** rule yields under pressure but restores itself: the magnetic timeline in Final Cut Pro resists gaps between clips, but clips can be forced apart before re-snapping on release. An **escapable** rule tolerates bypass that leaves a lasting result: indentation in VS Code enforces a tab size, but manual spacing can override it locally and the override persists. A **liftable** rule can be removed from enforcement entirely: spell-check in Word can be disabled for a document.

These two dimensions are orthogonal: any rigidity value can combine with any enforcement value. Desktop spreadsheet grids are (**fixed**, **escapable**): the grid logic itself is immutable, but paste-special operations and direct in-cell editing bypass dependencies and the results persist. The macOS Dock is (**negotiable**, **elastic**): size is adjustable within bounds, and magnification on hover yields then restores. The state couple thus generates a design space of 16 combinations.

3 Implications for Generative UI

The framework has three direct implications for genUI design.

GenUI must generate structural behavior, not just structural form. Current generative approaches focus on producing interface layouts and components [4]. The framework reveals that this is insufficient: for every rule a system generates, it must also specify that rule's rigidity and enforcement. A genUI system that deposits fully-formed content into a (**fixed**, **persistent**) structure produces an artifact users cannot modify or escape, which is consistent with the difficulties Cao et al. [3] observed when participants struggled to revise and explore within generated structures. One that generates (**negotiable**, **liftable**) structure would instead let users adjust generated constraints within bounds and selectively suspend them while exploring. The difference between a rigid, opaque genUI and a flexible, steerable one is not a matter of "how much AI" but of which state couples the system assigns to its generated rules.

Consider a genUI system that generates a data dashboard. If the column layout rules are (**fixed**, **persistent**), the user cannot rearrange columns or break out of the generated grid. If they are (**malleable**, **escapable**), the user can redefine the layout entirely, and local overrides (manually resizing a column, dragging a widget outside the grid) persist as lasting deviations. The structural form is the same; the structural behavior is entirely different.

Structural states should align with workflow phases. Creative and analytical workflows move through phases with different structural needs. Exploration benefits from lower rigidity (**negotiable** or **malleable**) with **liftable** enforcement: the user can adjust or redefine constraints and suspend those that get in the way. Execution benefits from higher rigidity (**fixed**) with **persistent** enforcement: constraints hold firm so that the user can focus on content without structural interference.

A genUI system aware of workflow phase could dynamically shift rule states rather than imposing a single structural configuration throughout. During ideation, layout constraints might be (**negotiable**, **liftable**): adjustable and removable. As the user commits to a direction, the system could tighten to (**fixed**, **elastic**): the layout becomes fixed but still yields elastically under pressure, providing guidance without rigidity. During finalization, (**fixed**, **persistent**) locks the structure. This progression from loose to tight structure mirrors how designers naturally work, but current genUI systems have no vocabulary to express it.

The framework provides a shared specification language. For genUI to be steerable, designers and users need a vocabulary to communicate structural intent to the system. The state couple (rigidity, enforcement) provides a compact, domain-independent notation. A designer could specify that a generated layout's column constraints should be (**negotiable**, **elastic**) and its typography rules (**fixed**, **persistent**), giving the system precise behavioral targets rather than vague directives like "make it flexible." A user could request that generated constraints be "liftable" without knowing anything about the framework, and the system could interpret this as a shift along the enforcement dimension. The vocabulary scales from expert specification to natural-language interaction.

4 Discussion and Conclusion

Structural Interaction suggests that HCI and design practice will need to evolve in at least two ways to support genUI.

First, the unit of design must expand. Current practice centers on designing interactions with content: selecting, dragging, editing domain objects. When structure is generated rather than hardcoded, designers must also specify how each structural rule should respond to user action. This requires new design tools: notation systems for expressing structural states, simulation environments for pre-viewing how a configuration behaves under user pressure, and evaluation methods that assess structural behavior independently of content quality.

Second, the relationship between user and structure must be renegotiated. In traditional interfaces, structure is the designer's province: users interact through structure, not with it. GenUI dissolves this boundary. If an LLM can generate and evolve structural rules [4], users need mechanisms to inspect, override, and author those rules, shifting along the rigidity dimension from **fixed** toward **authorable**. The framework clarifies that "user control over AI-generated interfaces" is not a single slider but at least two independent parameters per rule. A user who can override a generated constraint (**escapable**) but not redefine it (**fixed**) has a fundamentally different relationship to the system than one who can redefine constraints freely (**malleable**) but cannot escape them during interaction (**persistent**).

These claims require empirical validation. Other modalities or domains may reveal additional values or new dimensions. However, we believe the core argument is sound: as interfaces become generative, the organizational schemas through which users act become a primary design material. Structure is not merely something interfaces have; it is something interfaces do, and designing genUI through structural behavior opens new possibilities for interfaces that are simultaneously powerful, flexible, and human-centered.

References

- [1] Michel Beaudouin-Lafon. 2000. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proc. CHI '00*. ACM, 446–453.
- [2] Joanna Bergström and Kasper Hornbæk. 2025. DIRA: A model of the user interface. *Int. J. Hum.-Comput. Stud.* 193, C (2025), 14 pages.
- [3] Yining Cao, Yiyi Huang, Anh Truong, Hijung Valentina Shin, and Haijun Xia. 2025. Compositional Structures as Substrates for Human-AI Co-creation Environment. In *Proc. CHI '25*. ACM, Article 188.
- [4] Yining Cao, Peiling Jiang, and Haijun Xia. 2025. Generative and Malleable User Interfaces with Generative and Evolving Task-Driven Data Model. In *Proc. CHI '25*. ACM, Article 686.
- [5] Vincent Cavez, Caroline Appert, and Emmanuel Pietriga. 2024. Spreadsheets on Interactive Surfaces: Breaking through the Grid with the Pen. *ACM Trans. Comput.-Hum. Interact.* 31, 2, Article 16, 33 pages.
- [6] Clemens N. Klokmoose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proc. UIST '15*. ACM, 280–290.
- [7] Wendy E. Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive Systems. In *Proc. CHI '25*. ACM, Article 687.
- [8] Philip Tchernavskij. 2019. *Designing and Programming Malleable Software*. Ph.D. Dissertation. Université Paris-Saclay.
- [9] Jenifer Tidwell. 2010. *Designing Interfaces*. O'Reilly Media.
- [10] Vincent Cavez, Kashif Imteyaz, and Anne-Flore Cabouat. 2026. Structural Interaction: Shifting the Focus of User Interface Design. Under review.