Structural Interaction: Shifting the Focus of User Interface Design

Vincent Cavez
vcavez@stanford.edu
Stanford University
Stanford, USA

Kashif Imteyaz
imteyaz.k@northeastern.edu
Northeastern University
Boston, USA

Anne-Flore Cabouat
anne-flore.cabouat@inria.fr
Université Paris-Saclay, CNRS, Inria
Orsay, France

Abstract

User interfaces rely on rules that govern how users create, organize, and transform content. We call these rules structure. While structure enables functionality, its rigid enforcement often conflicts with user intentions, particularly in productivity and creative workflows. As generative systems increasingly produce and adapt structure on behalf of users, there is a clear need for a vocabulary to reason about structural behavior. We introduce Structural Interaction, a framework that makes structure a primary object of design. We model the user interface as a directed graph of elements and rules, and characterize rule behavior along two orthogonal dimensions: rigidity (how much a rule can be changed) and enforcement (how strictly it is enforced during interaction). Four values per dimension generate a 16-cell design space. Through two use cases, we show how the framework diagnoses structural limitations in existing interfaces and guides the design of solutions operating independently on each dimension.

Keywords

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

ACM Reference Format:
Vincent Cavez, Kashif Imteyaz, and Anne-Flore Cabouat. 2026. Structural Interaction: Shifting the Focus of User Interface Design. In . ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Productivity and creativity support tools, such as spreadsheets, text editors, design applications, and notation software, share a fundamental characteristic: they rely on rules that govern how users can create, organize, and transform content. A spreadsheet's grid, for instance, determines that data exists in cells addressable by row and column, enabling formulas that reference cells, sorts that preserve row integrity, and filters that reveal subsets. But these rules simultaneously enables and constrains [11]: the same grid becomes an obstacle when user intentions diverge from its logic, for instance to perform custom operations on values across the table. It is ultimately a question of balance. We call the set of such rules structure. As generative systems are now able to produce and adapt structure on behalf of users, and as users gain the ability to reshape interfaces through natural language and direct manipulation [4], there is a clear need for a shared vocabulary to reason about how much structure should yield, when, and under whose control, so that structural decisions can be made deliberately rather than by default.

Previous research has explored several concepts to reason about UI design, such as malleable interfaces [8], interaction substrates [9], and models that decompose UIs into constituent elements [2], but none provides a systematic way to characterize how rules — individually or as coherent groups — should behave during interaction.

This paper introduces *Structural Interaction*, a theoretical framework that places structure at the center of interaction design. We model the user interface as a directed graph of elements and rules, and characterize the behavior of rules 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.) Through two use cases in productivity and creative tools, we show how this characterization enables both the diagnosis of structural limitations in existing interfaces and the design of solutions. Just as direct manipulation shifted the focus to the objects users manipulate, Structural Interaction shifts the focus to the rules through which manipulation occurs.

2 Related Work

Understanding what makes a user interface work requires identifying its constituent parts. Beaudouin-Lafon's instrumental interaction introduced instruments as mediators between users and domain objects, and meta-instruments as mechanisms through which instruments act on other instruments [1]. Bergström and Hornbæk's DIRA model decomposes interfaces into Devices, Interaction Techniques, Representations, and Assemblies [2], where Assemblies, defined as what "organize[s] the representations and connect[s] interaction techniques to the computer," is conceptually closest to what we call structure. However, DIRA describes the compositional anatomy of interfaces without characterizing how assemblies should behave. Other work has explored making interfaces more flexible: Klokmose et *al.*'s Webstrates [8] and Tchernavskij's malleable software [10] demonstrate that users can reshape interface structure at runtime, but treat malleability as a platform-level property rather than a per-rule design parameter. Mackay and Beaudouin-Lafon's interaction substrates [9] go further by defining structured environments where users can adjust individual constraints through tweaking and create reusable configurations through templating. These operations address how much users can change a constraint, but the question of how a constraint responds when users push against it during interaction remains implicit rather than systematically characterized.

Recent work on AI-mediated creation makes this question more pressing. Cao, Jiang, and Xia propose generative interfaces driven

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

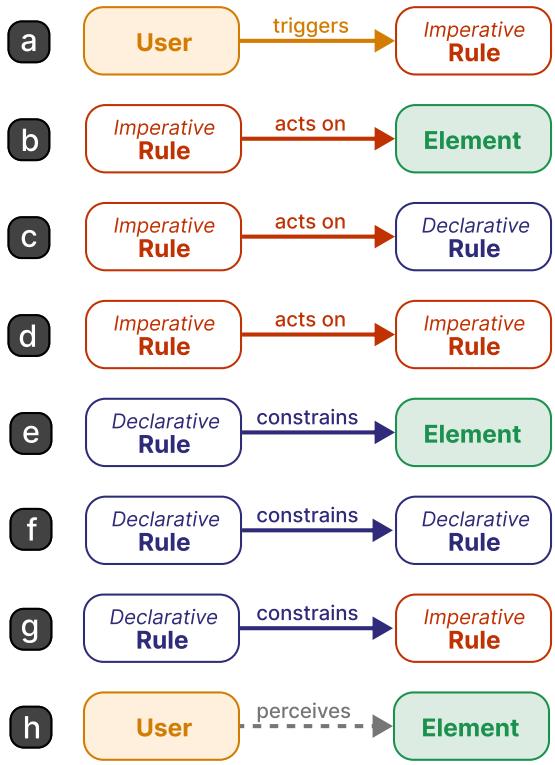

Figure 1: A directed graph model of a user interface and its eight connection types. The user triggers imperative rules (a). Imperative rules act on elements (b), on declarative rules (c), or on other imperative rules (d). Declarative rules constrain elements (e), other declarative rules (f), or imperative rules (g). The user perceives elements (h).

by task-driven data models that an LLM generates and evolves [4], 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 struggled to revise, iterate, and explore freely within the structures provided.

While research on user interface design increasingly recognizes that structure is central to making interfaces more adaptable, we still lack a shared definition of what structure is or how its behavior should be reasoned about. In this work, we articulate a definition of structure and introduce a conceptual framework to reason about how structural behavior shapes interaction.

3 Defining Structure

To define structure, we first represent the User Interface as a directed graph containing two types of nodes: *elements* and *rules*. Elements are the entities that users perceive. A spreadsheet cell, a bar on a musical staff, a layer in a compositing tool, and a node in a visual programming environment are all elements. 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 we define structure as the set of all rules governing the organization of elements.

In this view, rules encode all relationships and transformations that exist between elements or between other rules. We further identify two types of rules. *Declarative* rules are constraints that hold continuously and need no trigger: content rules such as "cell B1 always displays the value of A1 multiplied by two," presentation rules such as "all elements in a layer inherit that layer's opacity," and layout rules such as "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."

The edges of the graph are directional connectors. Eight kinds of connections are possible (Figure 1). The user triggers imperative rules (a). Imperative rules act on elements (b), declarative rules (c) or other imperative rules (d). For instance, a shortcut that toggles snap-to-grid is an imperative rule modifying the enforcement of a declarative rule (d). Declarative rules constrain elements (e), other declarative rules (f) or imperative rules (g): a read-only mode, for instance, is a declarative rule that prevents a set of imperative rules from executing. Finally, the user perceives elements (h), closing the interaction loop: the user triggers rules, rules act on elements and on other rules, and the user perceives the result.

4 Structural Interaction

Because this representation abstracts away device- and domain-specific details, it makes it possible to define and reason about structure consistently across diverse contexts. And because elements are always targets in the graph, never sources, every transformation in the interface ultimately acts on or through rules: users never interact with elements directly; they interact with structure. We call this shift in perspective *Structural Interaction*: instead of designing objects for users to manipulate, we design the rules through which manipulation occurs.

But designing rules individually would be impractical: in any structure, closely related rules tend to cluster into sub-structures that users experience as a unit. A spreadsheet's grid logic, for instance, is a sub-structure: a cluster of declarative and imperative rules governing how cells are created, addressed, referenced, and manipulated. These sub-structures sit at the center of the user's activity, and their behavior determines much of the interaction's quality. A poorly behaved isolated rule is a minor concern; a poorly behaved sub-structure compromises the tool. To reason about this behavior, we observe that the rules within a sub-structure tend to share a dominant behavioral state, which we describe along two independent dimensions: *Rigidity* and *Enforcement*.

.

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: no other rule can modify it (e.g., the email schema in Outlook requires a sender, a recipient, a subject, and a body). 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

Table 1: The 16-cell design space generated by the Rigidity and Enforcement dimensions, with examples of sub-structures from existing interfaces.

	Enforcement			
Rigidity	***persistent***	***elastic***	***escapable***	***liftable***
fixed	**Email structure** in Outlook/Gmail (schema is immutable, no field can be bypassed or removed)	**Element positioning** in Sibelius (layout rules are immutable, magnetic snapping, restores default alignment)	**Spreadsheet grid** in Excel (grid logic is immutable, paste-special, direct in-cell editing can bypass dependencies)	**Read-only mode** in Word (protection behavior is immutable, toggling it off lifts all editing constraints entirely)
negotiable	**Volume limiter** on iOS (maximum level is adjustable within a fixed range, limit blocks hard, never yields)	**Magnetic timeline** in Final Cut Pro (snap strength adjustable, clips resist gaps, re-snap on release)	**Indentation** in VS Code (tab size is adjustable within presets, manual spacing bypasses the rule momentarily)	**Spell-check** in Word (language and strictness are adjustable within presets, disableable entirely)
malleable	**Keyboard shortcuts** in Photoshop (shortcuts are entirely reassignable, once set, always active, never yields)	**Layout guides** in Illustrator (user creates guides, alignment constraints activate, release when elements move away)	**Slide layouts** in Keynote (layouts are entirely redefinable, manual repositioning overrides on a single slide)	**Auto-layout** in Figma (constraints are entirely redefinable, auto-layout removable entirely)
authorable	**Custom constraints** in Fusion 360 (user creates novel geometric constraints, solver blocks violations, never yields)	**Scroll-snap** in CSS (developer defines custom snap points, scrolling overshoots, snaps back on release)	**Macros** in Excel (user creates novel macros, Ctrl+Break can interrupt their execution)	**Filters** in Gmail (user creates novel filtering rules, disableable entirely)

a custom keyboard shortcut in Photoshop. An ***authorable*** rule goes further: the user can create rules that did not previously exist, as in a visual programming environment where users define new interaction behaviors from scratch.

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 momentary bypass: indentation in VS Code enforces a tab size, but manual spacing can override it locally. A ***liftable*** rule can be removed from enforcement temporarily or indefinitely: spell-check in Word can be disabled entirely for a document.

These two dimensions are orthogonal: any *Rigidity* value can combine with any *Enforcement* value. A (***malleable***, ***persistent***) rule describes a user-configured auto-save interval that, once set, never yields during interaction. The state couple thus generates a design space of 16 combinations. We illustrate this design space in Table 1 with examples of sub-structures from existing interfaces, each positioned according to its dominant state couple.

5 The Framework in Practice

To illustrate the framework's diagnostic and generative potential, we present two use cases that isolate complementary aspects of the state couple: the first involves a pure enforcement shift within a single sub-structure, while the second requires independent moves along both dimensions across two sub-structures.

5.1 Restoring Escapability: Spreadsheets on Tablets

The power of desktop spreadsheets lies in a specific structural configuration. The grid is a sub-structure whose dominant state is (***fixed***, ***escapable***): the grid logic itself is immutable (cells exist at row-column intersections, formulas reference addresses, sorting preserves row integrity), but nearly every rule tolerates momentary bypass. Paste-special operations that circumvent formula dependencies, and direct text manipulation within cells all constitute escape mechanisms. Users rarely notice this escapability because it is pervasive: what feels like "using a spreadsheet" is in fact continuously escaping the grid's fixed sub-structure at the micro-interaction level.

When spreadsheets migrated to tablet interfaces, touch input and simplified interaction models eliminated most escape mechanisms. The same rules now blocked hard where they previously yielded. Cavez et *al.* [5] documented the resulting usability problems. Our framework identifies the problem not as "tablets are harder to use" but as a shift of the grid's dominant state from (***fixed***, ***escapable***) to (***fixed***, ***persistent***) for specific rules governing character selection, cell boundary crossing, and content-based row manipulation. Cavez et *al.*'s solution, pen-based interactions enabling fluid character-level selection across cell boundaries, restored the ***escapable*** enforcement without altering the grid's ***fixed*** rigidity.

5.2 Relaxing Rigidity and Enforcement: Creative Expression in Score Writing

Commercial score-writing applications enable composers to produce high-quality scores following the Western musical notation. Most activities on these tools are dictated by the notation rules, and can be divided in two sub-structures with distinct dominant states. Musical rules (pitch spelling, rhythmic alignment, voice leading) form a sub-structure whose dominant state is (***fixed***, ***persistent***): they admit no deviation and do not yield during interaction. Positioning rules (element placement, bar spacing) form a sub-structure whose dominant state is (***fixed***, ***elastic***): magnetic snapping resists manual adjustment but yields under sustained pressure before restoring default alignment. This configuration hinders exploration. Composers wishing to sketch ideas, try alternatives, or work with partially formed musical thoughts must either work within the full

notation enforcement or abandon digital structure altogether by relying on paper [7].

Cavez et *al.* [6] addressed this by targeting each sub-structure independently. They shifted the rigidity of the musical rules sub-structure from ***fixed*** to ***negotiable*** within designated canvas zones: composers can create freeform areas where notation constraints relax, enabling handwritten sketches that coexist with structured notation. These zones persist indefinitely without forced reversion to full structure, and their content retains operational utility (playback, search, conversion). They shifted the enforcement of the positioning sub-structure from ***elastic*** to ***liftable***: elements move freely during pen manipulation, and alignment constraints are not automatically restored but reapplied by the composer when the passage is ready. The composer's experience of "structural relaxation" thus decomposes into two independent design moves, one per sub-structure, each addressable separately through the state couple.

6 Discussion

The two use cases illustrate that structural problems can be diagnosed precisely once rigidity and enforcement are distinguished. In Section 5.1, the grid's rigidity was never the issue: it remained fixed on both desktop and tablet. What changed was enforcement, from escapable to persistent, and the solution was to restore escapability. In Section 5.2, the solution required independent moves along both dimensions across two sub-structures. The framework thus helps avoid misdiagnoses: "this tool is too rigid" may conflate two independent properties requiring different solutions.

The framework complements existing models rather than replacing them. Where DIRA [2] decomposes interfaces into constituent parts, Structural Interaction characterizes how the organizational component should behave. Where interaction substrates [9] define structured environments that users can adjust, the state couple specifies how individual rules within those environments should respond during interaction. The diagnostic capacity also extends beyond our two cases: Cao et al.'s findings on premature convergence and fear of destructive edits in a co-creation environment [3] map directly onto enforcement states (persistent with no elastic or escapable alternative) and rigidity states (fixed where negotiable would better support exploration).

As a compact theoretical contribution, this framework calls for further development along several directions. Full-scale interface design projects that adopt the state couple as a first-class design parameter would test whether the 16-cell space is sufficient or whether additional values and dimensions are needed. Empirical studies could establish whether specific state couples predict measurable usability outcomes. We see this framework as one that should evolve through community engagement: designers applying it to new domains, researchers stress-testing its boundaries, and practitioners reporting where its vocabulary falls short. We hope the concepts introduced here provide a useful starting point for that collective effort.

7 Conclusion

We introduced Structural Interaction, a theoretical framework that places structure at the center of interaction design. By defining the User Interface as a graph of elements and rules and distinguishing two independent dimensions of rule behavior, rigidity (how much a rule can be changed) and enforcement (how strictly it is enforced during interaction), the framework provides designers with a vocabulary for reasoning about when structure should constrain, adapt, or yield. Through two use cases we showed that structural problems that appear similar on the surface decompose into distinct combinations of rigidity and enforcement, each requiring a different design response. The 16-cell design space generated by the two dimensions is deliberately compact: it aims to be large enough to capture meaningful distinctions and small enough to remain usable as a design tool. Structure is not merely something interfaces have; it is something interfaces do. Designing interaction through the lens of structural behavior opens new possibilities for interfaces that are simultaneously powerful and flexible.

References

[1] Michel Beaudouin-Lafon. 2000. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands) *(CHI '00)*. Association for Computing Machinery, New York, NY, USA, 446–453. doi:10.1145/332040.332473

[2] Joanna Bergström and Kasper Hornbæk. 2025. DIRA: A model of the user interface. *Int. J. Hum.-Comput. Stud.* 193, C (Jan. 2025), 14 pages. doi:10.1016/j.ijhcs.2024.103381

[3] Yining Cao, Yiyi Huang, Anh Truong, Hijung Valentina Shin, and Haijun Xia. 2025. Compositional Structures as Substrates for Human-AI Co-creation Environment: A Design Approach and A Case Study. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 188, 25 pages. doi:10.1145/3706598.3713401

[4] Yining Cao, Peiling Jiang, and Haijun Xia. 2025. Generative and Malleable User Interfaces with Generative and Evolving Task-Driven Data Model. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 686, 20 pages. doi:10.1145/3706598.3713285

[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 (Jan. 2024), 33 pages. doi:10.1145/3630097

[6] Vincent Cavez, Catherine Letondal, Caroline Appert, and Emmanuel Pietriga. 2025. EuterPen: Unleashing Creative Expression in Music Score Writing. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 760, 16 pages. doi:10.1145/3706598.3713488

[7] Vincent Cavez, Catherine Letondal, Emmanuel Pietriga, and Caroline Appert. 2024. Challenges of Music Score Writing and the Potentials of Interactive Surfaces. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 728, 16 pages. doi:10.1145/3613904.3642079

[8] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 280–290. doi:10.1145/2807442.2807446

[9] Wendy E. Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive Systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 687, 16 pages. doi:10.1145/3706598.3714006

[10] Philip Tchernavskij. 2019. *Designing and Programming Malleable Software.* Ph. D. Dissertation. http://www.theses.fr/2019SACLS499

[11] Jenifer Tidwell. 2010. *Designing Interfaces.* O'Reilly Media, Inc.