Substrates via Accessibility

February 2026

Orion Reed
me@orionreed.com

Chris Shank
chris.shank.23@gmail.com

Abstract

The social model of disability holds that disability is produced by environments, not by bodies — that when software fails to accommodate a person's needs, the environment is disabling them. We argue that the dominant organization of computing produces such interaction mismatches structurally: the commodity form of software requires a standardized consumer, and everyone whose needs diverge from this imagined subject is disabled by the software environment. The substrates research community has identified properties — disclosure, composability, malleability — that would address this, but the commodity form actively selects against these properties because they dissolve the boundaries on which exchange value depends. This raises a question: how can infrastructure organized against the grain of commodity logic be made durable? We find a precedent in accessibility infrastructure — sustained not by patronage but by the organized counter-power of disability rights movements. Through Allio, a system for cross-application augmentation built on accessibility APIs, we illustrate both what this foundation makes possible and where its limits lie. We argue that the accessibility case reframes the substrates programme's central challenge as political-economic: the question of who sustains substrate infrastructure, and through what social mechanisms, is as important as the question of what properties it should have.

A web version of this paper with accompanying demo videos is available at folkjs.org/substrates-2026

1. Introduction

In 1976, the Union of the Physically Impaired Against Segregation argued that it is society, not impairment, that disables people — that disability is imposed by environments which unnecessarily isolate and exclude. The *social model of disability*, as Oliver later formalized it, draws a distinction between *impairment* — a bodily or cognitive characteristic — and *disability* — the barriers that an environment places in a person's way. [1] Disability is an *interaction mismatch*, produced by environments that fail to account for the people who must navigate them [2]. A person with a broken arm, a bus driver who cannot look at a screen, and a worker in a loud factory are all, in their context, disabled by an environment that assumes two free hands, constant visual attention, or a quiet room.

Software produces interaction mismatches at enormous scale. The dominant organization of computing — mass-produced applications, each enclosing its own data, tools, and representations — serves the *unmarked user*, the imagined subject of the dominant and most profitable demographic, while systematically failing the long tail of everyone else [3]. We believe this is not a flaw in particular applications but a structural feature of the model: siloed, one-size-fits-all software, sold and distributed as discrete units, reflecting the priorities of its producers. What a person needs but the producer did not anticipate is, in the precise sense of the social model, *disabled* — not because the need is illegitimate but because the environment fails to accommodate it.

If the problem is produced by how software is organized rather than a deficiency in the people who use it, then an intervention at the root requires a change in the organization of software itself. The substrates research community has explored what such a reorganization might look like: *computational materials* that disclose their internal structure, that are composable and can be operated on by instruments from any source, and that blur the distinction between programming and use. In this vision, the fundamental unit of software shifts from the *product* (an application, bounded and owned by a producer) to the *material* (a computational medium that exists independently of any particular application and can be worked on and adapted across many contexts of use). But one cannot have rich computational materials without a structural separation between the materials and the tools — having wood and nails means nothing if you can only use the nails from select suppliers with business ties to the wood vendor. A shift to material-centric computing is a reorganization not just of software's properties but of its ontology: what the basic objects of computing are, and how social and economic processes produce them.

We argue that disability studies provides both an analytical framework and a political precedent for this programme. The social model's analysis — that interaction mismatches are produced by environments, not by bodies — parallels the condition substrates research diagnoses, and through political organizing, litigation, and standards work, disability rights movements produced the only infrastructure in contemporary computing that organizes software by *what things are* rather than who made them: accessibility infrastructure present on every major consumer operating system.

In this paper we examine accessibility infrastructure as a foundation for substrate-oriented work, illustrate what it makes possible through Allio — a system for cross-application augmentation built on this foundation — and

argue that the accessibility case reframes the substrates programme's central challenge as political-economic.

2. Software as Commodity

The substrates community has identified desirable properties that software largely does not have: programming and use on a spectrum rather than distinct [4], no technical distinction between manipulating tools and data [5], disclosure that makes the path from use to modification "reasonably direct" [6]. The dominant application model denies all of these: source code is hidden and often illegal to modify, data formats are largely proprietary, internal structure is opaque. But *why* does software take this form?

Through the personal computing era and into the present, the application boundary can be understood as the *commodity boundary* — the unit that can be named, sold, installed, updated, and owned by a producer. An application must be a discrete unit that can be exchanged; it requires a producer-consumer relation between developer and user; and because software can be copied at zero marginal cost, it requires artificial scarcity mechanisms — proprietary formats, restricted APIs, platform lock-in, source secrecy — to sustain exchange value. This is not a neutral technical arrangement but a self-reinforcing one. As Tchernavskij demonstrates, Parnas's invention of information hiding was motivated by a specific mode of production: top-down, centralized management of large disconnected teams in an industrial design process. [7] This technique, devised for commodity production, became encapsulation in object-oriented programming and embedded itself in compilers, frameworks, and IDEs. The mode of production inscribes itself in the tools; the tools reproduce the mode of production.

The social model helps clarify what this feedback loop produces at the level of the user. Commodity software must address a market which requires a *generalizable* consumer. The technical enclosure of the application — its sealed data, hidden source, proprietary formats — is simultaneously an enclosure of the user within a fixed set of anticipated needs. The further a person's actual needs diverge from the producer's model, the more the software environment disables them. This is not a failure of empathy on the part of individual developers; it is a structural consequence of the commodity form, which must produce Costanza-Chock's *unmarked user* — the default subject of the dominant and most profitable demographic — as its standard subject in order to produce software as a standard commodity. [3] The properties substrates research seeks — disclosure, composability, malleability — are not merely absent from contemporary software but are properties that the commodity form structurally selects *against*, because they dissolve the boundaries on which exchange value depends.

Church and Samosir observe that "software can only live in tension with the epistemic power of its surroundings for brief periods of time, and at the expense of significant investments of energy by the people building it." [8] This captures the trajectory of every designed substrate: HyperCard, OpenDoc, Smalltalk environments, and contemporary research prototypes persist only as long as someone sustains them against the prevailing logic of commodity software production. Interoperability is a *negative externality* of the commodity form — for any individual software producer, making their product work well with competitors' reduces lock-in and thus profit. A 1987 Stanford paper on legal protection for product standards demonstrated that IP protection *strengthens the incentive* to develop incompatible standards, and the reward for successful incompatibility *increases with market share*. [9] We believe this commodity logic, now embedded in our tools, techniques, and infrastructure, is a core obstacle to the substrates research programme.

3. Accessibility Infrastructure

The earliest accessibility infrastructure was adversarial: in text-based systems of the 1970s, screen readers extracted text from video card buffers without the application's knowledge; when graphical interfaces arrived in the 1980s, they resorted to patching rendering systems and display drivers to reverse-engineer what was on screen. [10] These techniques anticipate, in crude form, the kind of cross-application instrumentation that substrates research envisions with far richer infrastructure.

Now every major operating system provides an accessibility framework: macOS's Accessibility API, Windows's UI Automation, GNOME's AT-SPI over D-Bus, and corresponding mobile frameworks on iOS and Android. Applications expose their interactive elements — buttons, text fields, sliders, menus, tables — as a tree of nodes described using a shared vocabulary of roles, states, attributes, and actions. The tree is queryable at runtime. Elements can be discovered, inspected, and — to varying degrees — acted upon by external processes.

Users whose work does not fit neatly into application boundaries must perform constant, silent translation between incommensurable descriptions of their own activity. Other cross-application infrastructure does not solve this: the filesystem addresses data at rest, platform automation like AppleScript or D-Bus exposes only what individual applications choose to offer, and protocols like SMTP or HTTP are channels *between* bounded systems — they enable communication across application boundaries without dissolving them. The accessibility tree does something structurally different. It provides a unified description layer *across* applications — a uniform, live description of the interactive surface of the computer as a single addressable space, using a shared vocabulary: a button is a button regardless of which process produced it, which language it was written in, or which vendor shipped it. Applications, in this representation, are incidental containers — a process

ID, a window frame — not fundamental units of organization.

This infrastructure was not designed for interoperability, software composition, or end-user modification. It was designed so that blind, visually impaired, and motor-impaired people could use computers. A screen reader must be able to narrate *any* application; a switch control system must be able to act on *any* interactive element. Universality was non-negotiable because anything less would leave disabled people at the mercy of each individual developer's willingness to cooperate and coordinate. In practice, compliance is uneven: Electron applications often expose impoverished trees, custom-drawn interfaces may expose almost nothing, and web applications frequently produce broken ARIA markup. The result is what Dolmage calls *retrofitting*: applications are built first, and then an accessibility layer is imposed as a corrective, after the fact. [11]

Yet accessibility infrastructure has proven remarkably durable, surviving the desktop-to-mobile transition, the native-to-web transition, and multiple platform generations. It is sustained not by the continued energy of researchers or the continued funding of a particular project, but by a self-reproducing political and legal structure: organized constituencies, legal instruments (ADA, Section 508, EU accessibility directives), institutional obligations on platform vendors, and the ongoing threat of litigation.

Against the definitions debated at Substrates-25, the accessibility tree provides a *foundation* rather than a substrate. It satisfies none of Edwards' criteria — it is not a programming system, has no persistent store, provides no direct manipulation. It partially meets Klokmose's principles [12]: cross-application manipulability and layered semantics, but not transclusion, appropriation in use, or history. It has a relationship to Basman's concept of disclosure [6] — the accessibility tree is literally a disclosure mechanism, making the internal structure of applications legible to external processes — but it discloses only interface metadata, not structured data or computational operations. It provides an ontological foundation without the material depth. These limitations are traces of the specific political demand that produced the infrastructure: the demand for machine-readable interfaces *for assistive technology*, not for general-purpose composition. The technology is shaped by the scope of the struggle that won it.

Gobert frames a productive tension between *modern* and *postmodern* approaches to substrates: modernism proposes replacing existing systems with something designed from the ground up; postmodernism accepts the computing world as it is and designs within its constraints. [13] Kell's *compatibilism* — "embracing the existing code that already does the job" rather than building facsimiles — is one version of the postmodern position. [14] But what neither framing fully addresses is the question of *durability*: what socio-economic conditions must hold for substrate infrastructure to persist once its creators' energy dissipates? The accessibility case, we argue, demonstrates that durability requires forces external to commodity logic.

4. Reflections from Allio

Allio is an experimental system that materializes a proto-substrate (a system which lacks many properties of a fully-realized substrate, but which shifts the ontology from app-centric to material-centric) on top of accessibility infrastructure. Where the traditional model pairs a specific assistive technology to the accessibility API in a one-to-one relationship — a screen reader consuming the tree, a switch control acting through it — Allio exposes the tree as a general-purpose surface on which *arbitrary* augmentations can be built. In Gobert's terms, this is a *postmodern* approach — building on found infrastructure [13] — animated by the structural analysis of *why* this particular infrastructure exists and what political conditions sustain it.

Allio maintains a synchronized cache of accessibility state from the operating system and exposes it to browser-based clients, so that web technologies serve as the authoring environment for augmentations that operate on native applications. The closest analogue is the browser's DOM — a queryable, semantically described tree of interactive elements — but unlike the DOM, which is sandboxed per origin, Allio's tree spans all open applications, exposing their structure as a single unified space. Because the tree exposes bounding boxes for every element, Allio can render web-based overlays on top of the desktop, layered to respect window depth ordering — augmenting applications from the outside.

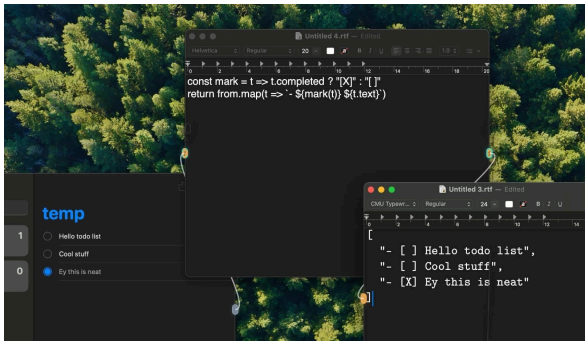

Figure 1: A desktop overlay using Allio to turn the desktop into a live node-and-wire interface.

We have built several prototypes on top of Allio including a find-and-replace feature which works for any text field, a CSS-like query language to extract data from the tree and pipe it between different applications, and a physics simulation which incorporates window geometry into the simulation. In each case, UI affordances are rendered on top of, around the edges of, and behind application windows so as to match the native window manager's notion of depth order and application focus.

We have demonstrated Allio to several audiences — researchers, developers, and people with no technical background — and their reactions have been remarkably

consistent. People respond with genuine surprise at the *mundanity* of what they see. Cross-application find-and-replace is not exotic functionality — every text editor has find-and-replace. The surprise is at familiar capability appearing outside the container it was supposed to be confined to. People have been trained to understand specific text search tools as something that belongs to an application (a product), not something they do with text (a material). That is, the constraints imposed by the application model come to appear not as design choices but as facts about how computers work. The commodity form shapes not just what software provides but what users understand themselves to need.

Developers are often more surprised than non-technical users. Where a non-developer reacts with curiosity — "I didn't know you could do that" — developers react with disbelief: process isolation, sandboxing, and API boundaries are understood as architectural necessities, not contingent arrangements. Technical education deepens the naturalization rather than providing critical distance from it. The Parnas feedback loop — mode of production inscribing itself in tools, tools reproducing the mode of production — operates at the level of individual cognition as well as industry structure.

The ease with which people denaturalize their assumptions is, we think, an argument for the postmodern approach: augmenting familiar environments builds a legibility and a constituency that designed-from-scratch substrates cannot. But constituency at the level of experience does not automatically translate into change at the level of infrastructure. The limits catalogued below are not ones that a compelling demo can dissolve.

Building Allio has made concrete the technical limits of this foundation. Accessibility trees are predominantly pull-based — consumers must poll for state, and events are inconsistent across platforms — and biased toward reading over writing, since screen readers need to observe interfaces, not modify them. These first two limitations likely reflect the performance constraints and architectural assumptions of the era in which the APIs were designed. The deeper gap is semantic: the tree describes elements generically — a text field is "a text field containing these characters," not "a Markdown editor with these operations." The foundation provides an ontology of interface but not of content or computation. This limitation does trace to the scope of the constituency that fought for the infrastructure: the demand was for machine-readable interfaces for assistive technology, not for general-purpose composition, and the technology was shaped accordingly.

5. Accessibility is Malleability

The substrates research community has been exploring how to give people agency over the computational materials of their work. Disability rights movements have been fighting for this too, under different names and through different means, for far longer. But these are not identical programmes, and conflating them risks the appropriation that Shew warns against under the name *technoableism* — the assumption that technology is a "solution" for disability, that disabled people await being "fixed." [15] If research builds on accessibility infrastructure without centering the people whose political struggle produced it, it risks extracting value from hard-won infrastructure for purposes its creators had no part in defining.

This risk must be confronted directly. Allio repurposes accessibility infrastructure for general augmentation, and the honest assessment is that this sits in an ambiguous position. We think two conditions distinguish engagement from extraction. First, general-purpose use of accessibility trees creates a broader constituency with an interest in the trees' richness and reliability — if more tools depend on the tree, more pressure exists to ensure applications expose complete and accurate trees, which directly benefits assistive technology users. Second, and more importantly, broadening the constituency must not displace the specific needs of disabled users. The disability rights demand for accessible software is not a special case of the substrates demand for malleable software; it is a prior and independent demand with its own political legitimacy. Substrates researchers who build on this foundation have, at minimum, an obligation not to treat accessibility compliance as a technical convenience while ignoring the access needs that motivated it.

What disability studies contributes to the substrates programme is not a metaphor but a structural analysis and a political precedent. The social model's core insight — that disability is produced by environments, not by bodies — is a claim about the relationship between people and the systems they inhabit. The demand that software be accessible *is* a demand that it be malleable: malleable enough to accommodate the diversity of bodies, senses, cognitive styles, and interaction modalities that the commodity form, through its production of the unmarked user, treats as edge cases. Malleability, in this framing, is not an abstract design goal but an access need — one that the commodity form of software systematically fails to meet.

The political precedent is equally important. Designed substrates often die when their creators' energy dissipates, because they exist in tension with the logic that organizes the rest of computing. Church's question — where might alternative sources of patronage come from? — is one the accessibility case answers concretely. [8] Accessibility infrastructure is not sustained by patronage. It is sustained by *counter-power* — organized constituencies capable of imposing costs on non-compliance — that won legal mandates, creating a self-reproducing structure of law and institutional obligation that persists independently of any particular researcher, company, or funding cycle. This demonstrates that computing infrastructure organized against the grain of capital *can* be made durable, if the political conditions are met.

This reframes a central challenge of the substrates programme. The technical problems — reactivity, composability, disclosure, structured I/O — are real and difficult. But the accessibility case suggests that the deeper challenge is political-economic: how to produce and sustain infrastructure that is economically selected against. The current demand for AI agents to interact with arbitrary applications lends new urgency to this question: agent frameworks increasingly use accessibility trees as their primary interface to desktop software, creating for the first time a *market incentive* to enrich exactly the infrastructure that was previously sustained only against market logic. Whether this convergence deepens the foundation for everyone or produces new enclosures built on decades of disability rights organizing is a question where the substrates community's values are directly at stake.

This does not mean that substrate researchers must become political organizers. It means that the question of *who sustains* substrate infrastructure and *through what social mechanisms* is as important as the question of what properties it should have. The accessibility case offers both a model and a caution: a model of infrastructure sustained by organized counter-power, and a caution that such infrastructure will reflect the specific scope and demands of the constituency that fought for it.

6. Conclusion

The substrates research programme has produced compelling visions of what computing could be and has built working prototypes that demonstrate these properties within their own boundaries. What it has not yet fully reckoned with is the question of *durability*: how substrate infrastructure can persist under the socio-economic conditions of commodity software production. The accessibility case provides a precedent — not a template, but a demonstration that computing infrastructure organized against the grain of the commodity form can endure, if it is embedded in a self-reproducing structure of organized political power.

Allio demonstrates that this foundation can be materialized into a proto-substrate for cross-application augmentation — and that when it is, the dissolution of application boundaries produces genuine surprise, which is evidence of how deeply the commodity form has shaped not just what software provides but what users understand themselves to need. Allio also demonstrates the limits: the technical gaps it encounters trace variously to historical design constraints and to the scope of the political demand that produced the infrastructure. Deepening this foundation will require both technical work and a broadening of the constituency that sustains it.

References

[1] M. Oliver, "The social model of disability: thirty years on," *Disability & Society*, vol. 28, no. 7, pp. 1024–1026, Oct. 2013, doi: 10.1080/09687599.2013.818773.

[2] Microsoft Design, "Inclusive Microsoft Design." [Online]. Available: https://inclusive.microsoft.design/tools-and-activities/Inclusive101Guidebook.pdf

[3] S. Costanza-Chock, *Design Justice: Community-Led Practices to Build the Worlds We Need*. The MIT Press, 2020. doi: 10.7551/mitpress/12255.001.0001.

[4] J. Edwards, "Substrate vision statement," June 2025, [Online]. Available: https://software-substrates.github.io/proceedings/2025/statements/Paper%203%20-%20Jonathan%20Edwards%20-%20Substrate%20vision%20statement.pdf

[5] M. Beaudouin-Lafon, "Beyond Applications: Interaction Substrates and Instruments," in *Proceedings of the 34th Conference on l'Interaction Humain-Machine*, in IHM '23. New York, NY, USA: Association for Computing Machinery, May 2023, pp. 1–15. doi: 10.1145/3583961.3583968.

[6] A. Basman, "To disclosable computing through concrete abstractions," June 2025, [Online]. Available: https://software-substrates.github.io/proceedings/2025/statements/Paper%208%20-%20Antranig%20Basman%20-%20To%20disclosable%20computing%20through%20concrete%20abstractions.pdf

[7] P. Tchernavskij, "Designing and Programming Malleable Software," phdthesis, 2019.

[8] L. Church and W. Samosir, "Software as a socio-technical coral," June 2025, [Online]. Available: https://software-substrates.github.io/proceedings/2025/statements/Paper%2017%20-%20Luke%20Church%20and%20William%20Samosir%20-%20Software%20as%20a%20socio-technical%20coral.pdf

[9] P. S. Menell, "Tailoring Legal Protection for Computer Software," *Stanford Law Review*, vol. 39, no. 6, p. 1329, July 1987, doi: 10.2307/1228849.

[10] AEGIS Consortium, "AEGIS OAF and high-level architecture." Accessed: Feb. 20, 2026. [Online]. Available: https://web.archive.org/web/20201229173652/https://www.aegis-project.eu/images/docs/AEGIS_D1.2.1_final-revised_2nd_Annual_Review.pdf

[11] J. Dolmage, "Mapping composition: Inviting disability in the front door," *Disability and the teaching of writing: A critical sourcebook*, pp. 14–27, 2008.

[12] C. N. Klokmose, "Substrates: From Webstrates and Beyond," June 2025, [Online]. Available: https://software-substrates.github.io/proceedings/2025/statements/Paper%205%20-

%20Clemens%20Klokmose%20-%20Substrates,%20From%20Webstrates%20and%20Beyond.pdf

[13] C. Gobert, "Designing postmodern substrate architectures," June 2025, [Online]. Available: https://software-substrates.github.io/proceedings/2025/statements/Paper%2011%20-%20Camille%20Gobert%20-%20Designing%20postmodern%20substrate%20architectures.pdf

[14] S. Kell, "substratus unicus," [Online]. Available: https://www.humprog.org/~stephen/research/reports/kell25substratus.pdf

[15] A. Shew, *Against technoableism: rethinking who needs improvement*, Norton paperback. in Norton shorts. New York, NY: W. W. Norton & Company, 2024.