Substrates: From Webstrates and Beyond

Clemens Nylandsted Klokmose
Aarhus University
Aarhus, Denmark
clemens@cs.au.dk

Abstract

In this position paper, I present my perspective on substrates with six non-exhaustive principles, grounded in my work and shaping future directions.

Keywords: Substrates, Webstrates, instrumental interaction, software architecture, transclusion

In my own research, the concept of *substrates* started to emerge in discussions with Michel Beaudouin-Lafon, Wendy Mackay, and James Eagan in the early 10s. Michel had, ten years earlier, written a paper on an alternative interaction model to Shneiderman's direct manipulation [14] that he called *instrumental interaction* [2]. In instrumental interaction, interaction is conceptualised as users manipulating domain objects mediated through *interaction instruments*. These instruments are described as two-way transducers between the user and domain objects, encapsulating commands that can be performed on those objects. The scrollbar is given as an example of something conventional that can be thought of in terms of instruments. Michel developed a set of properties to evaluate existing UIs through the lens of instruments and to generatively design new instruments. I started collaborating with Michel during my PhD studies around 2006, and I wanted to build a software system where the interaction architecture was based on instrumental interaction. Users would not interact with applications per se, but rather with dynamic collections of instruments tailored to the task at hand and adapted to the specific device the user was using. The ideal was that software would evolve with its user, and instruments could be picked and chosen based on skill, preferences, and the context of the task.

We wrote a paper in which we proposed an instrument-based alternative to Model-View-Controller (MVC) called VIGO (Views, Instruments, Governors, and Objects) [8]. In this model, users manipulated objects, which represented pure state, through instruments. Views rendered these objects and provided visual references that instruments could use for lookup. *Governors* were associated with objects and embodied the rules and consequences of interacting with them. One example we used was the Othello game. The game board was represented as objects consisting of the board itself and the pieces. A move instrument was used to manipulate the pieces, but it did not encode the game's rules. Instead, it queried available governors for valid moves, and a governor would compute the consequences of a move—such as flipping the opponent's pieces. The system was distributed, so governors could run locally, on a server, or on another device. Since the board was just a surface in the system, a drawing instrument could draw on it, just as it could on any other surface. However, we ended up violating a key principle of instrumental interaction:

> *[I]nstruments are themselves potential objects of interest. This is indeed the case in real life, when the focus of attention shifts from the object being manipulated to the tool used to manipulate it. For example a pencil is a writing instrument and the domain object is the text being written. When the lead breaks, the focus shifts to a new instrument, a pencil sharpener, which operates on the shifted domain object, the pencil lead. The focus may even shift to the pen sharpener, if we need a screwdriver to fix it. Such "meta-instruments" (instruments that operate on instruments) are not only useful for "fixing" instruments, but can also be used to organize instruments in the workspace, e.g. a toolbox, or to tailor instruments to particular tasks, e.g. turning a power-drill into a power-saw.* [2, pp.449]

What this means for a software architecture, is that manipulating instruments shouldn't be different from manipulating objects of interest. In fact, whether something is an object of interest or an instrument should be a phenomenon that emerges through use. This is how things work in the physical world—you can hammer on a hammer—but it feels strangely alien in the way we use GUIs.

In discussing this, the notion of *substrates* began to emerge. We wanted a software construct that could function as both an instrument and a domain object depending on use, and instruments could be used to manipulate other instruments as well[1]. This leads to what I propose as **substrates principle 1**:

[1]Wendy Mackay and Michel Beaudouin-Lafon have a recent paper on the conceptual model for substrates [12]. A discussion of how that aligns with the principles I outline here is a topic for another time.

There should be no technical distinction between manipulating your tools and manipulating your data.

Both in our work with VIGO and later experiments with the Shared Substance platform [6], real-time syncing of data across devices was essential. We wanted it to be the norm that users could work across their available devices, and be able to collaborate whenever it is necessary.

This leads to **substrates principle 2**: *All user content, be it tools or data, should be manipulable across devices and between users.*

Webstrates [9] (Web+substrates) is a pragmatic attempt to realise these first two principles in a working software architecture on the Web. Webstrates makes the document object model (DOM) of web pages the basic substrate for building collaborative and malleable software. We do this by synchronizing and persisting changes to the DOM using an operational transformation framework. In its simplest form, developers can build software simply by editing the DOM through the browser's developer tools, including modifications to embedded JavaScript code. In the Webstrates paper, we state that substrates are software artifacts that can be both applications and documents depending on their use. We demonstrate this by leveraging transclusion through iframes to establish a document—application-like relationship between webstrates, for instance, between a slideshow document and a slide editor. While editing a slideshow through the slide editor, another user could technically be editing the slide editor itself using the same mechanisms of transclusion. In Webstrates, software composition happens through transclusion as well, and it turns out to be powerful that, e.g., a software library is just a webstrate itself with the affordances that entails.

This leads to **substrates principle 3**: *Substrates should support composability through transclusion, both for interaction in use and for software construction.*

Using the browser's developer tools is not a scalable way of developing software, and we have built iterations of in-browser authoring environments on top of Webstrates [4, 13] as well as mechanisms for integrating with conventional IDEs. We have experimented with a new declarative programming model to allow for live and collaboratively reprogrammable software [5]. We have shown how this makes it possible to, e.g., create highly tailorable video conferencing tools that can even be reprogrammed live during a meeting [7]. At the same time, we have yet to succeed in creating an environment where programming is not conceptually detached from use. Even for experienced programmers, there remains a significant gap between using software and changing it.

From this follows **substrates principle 4**: *Users should be able to change or appropriate their software in the course of use, without having to adopt a programmer's mindset removed from their task domain.*

With Webstrates, we have explored the idea of layers of substrates. The base level being the DOM, but on top we have built layers for, e.g., video editing, data visualisation, and recently mixed-reality interfaces.

A webstrate can contain multiple layers, and a tool rendering videos from a Videostrates [11] layer can operate without knowledge of other layers, such as a Dashspace [3] mixed-reality data visualization. While some tools understand and leverage the semantics of a specific layer, others may manipulate its underlying data without awareness of the interpretations added at higher layers. With MyWebstrates [10], we rearchitected Webstrates as local-first software, and introduced a *data substrate* beneath the DOM substrate based on the Automerge CRDT[2]. This allowed for interoperability beyond Webstrates with software capable of interacting with Automerge.

This leads to **substrates principle 5**: *Substrates are layered, with each layer exposing different semantics and affordances, enabling tools to interact with the same data at different levels of abstraction.*

Finally, substrates must carry history. Access to edit history and powerful versioning tools is still largely confined to specific applications or text-based domains like programming. In Webstrates, we built on operational transformation, which inherently relies on maintaining a history of edits to the document. We have used this history to support primitive versioning, and in a malleable system like Webstrates, the ability to roll back changes is essential, especially when things break. MyWebstrates builds on Automerge, which has more sophisticated versioning support such as branching and merging. History is also *interaction history*, which currently remains confined within application silos, limiting its utility. Yet, as demonstrated by Battut et al. [1], integrating such histories across applications can empower users to better understand, organize, and resume their work.

So lastly, **substrates principle 6:** *Substrates should make edit and interaction history a first-class resource that can be reasoned about, combined across substrates, and used generatively in users' work.*

The above six principles are not exhaustive, but they reflect the kinds of questions and ideals that guide my work: systems that blur the line between tool and content, user and developer; and where collaboration and malleability are the norm, not the exception. These principles are not prescriptions, but openings for discussions at the workshop.

References

[1] Alexandre Battut, Kevin Ratovo, and Michel Beaudouin-Lafon. 2025. OneTrace: Improving Event Recall and Coordination With Cross-Application Interaction Histories. *International Journal of Human–Computer Interaction* 41, 5 (2025), 3241–3258.

[2] https://automerge.org

[2] 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*. 446–453.

[3] Marcel Borowski, Peter WS Butcher, Janus Bager Kristensen, Jonas Oxenbøll Petersen, Panagiotis D Ritsos, Clemens N Klokmose, and Niklas Elmqvist. 2025. DashSpace: A Live Collaborative Platform for Immersive and Ubiquitous Analytics. *IEEE Transactions on Visualization and Computer Graphics* (2025).

[4] Marcel Borowski, Janus Bager Kristensen, Rolf Bagge, and Clemens Nylandsted Klokmose. 2021. Codestrates v2: A development platform for Webstrates. (2021).

[5] Marcel Borowski, Luke Murray, Rolf Bagge, Janus Bager Kristensen, Arvind Satyanarayan, and Clemens Nylandsted Klokmose. 2022. Varv: Reprogrammable interactive software as a declarative data structure. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–20.

[6] Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. 2011. Shared substance: developing flexible multi-surface applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3383–3392.

[7] Jens Emil Sloth Grønbæk, Marcel Borowski, Eve Hoggan, Wendy E Mackay, Michel Beaudouin-Lafon, and Clemens Nylandsted Klokmose. 2023. Mirrorverse: Live tailoring of video conferencing interfaces. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[8] Clemens Nylandsted Klokmose and Michel Beaudouin-Lafon. 2009. VIGO: instrumental interaction in multi-surface environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 869–878.

[9] 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*. 280–290.

[10] Clemens Nylandsted Klokmose, James R Eagan, and Peter van Hardenberg. 2024. MyWebstrates: Webstrates as local-first software. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–12.

[11] Clemens N Klokmose, Christian Remy, Janus Bager Kristensen, Rolf Bagge, Michel Beaudouin-Lafon, and Wendy Mackay. 2019. Videostrates: Collaborative, distributed and programmable video manipulation. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 233–247.

[12] Wendy E. Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive Systems. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '25)* (Yokohama, Japan). ACM, New York, NY, USA, 1–16. doi:10.1145/3706598.3714006

[13] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R Eagan, and Clemens N Klokmose. 2017. Codestrates: Literate computing with webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 715–725.

[14] Ben Shneiderman. 1983. Direct manipulation: A step beyond programming languages. *Computer* 16, 08 (1983), 57–69.

*********************** PAPER 5 ***********************
AUTHORS: Clemens Nylandsted Klokmose
TITLE: Substrates: From Webstrates and Beyond

++++++++++ REVIEW 1 (Gilad Bracha) +++++++++
I am in violent agreement with the principles elucidated here. I note that your use of "substrate" seems to equate to "live document". Some thoughts:

a. "There should be no technical distinction between manipulating
your tools and manipulating your data." Yes. From my perspective, this is closely related to Smalltalk's "everything is an object" mantra. A good example of this is the Morphic UI, first developed in Self and later widely deployed via Squeak Smalltalk. However, this is in some tension with principle 5, "Substrates are layered, with each layer exposing different semantics and affordances, enabling tools to interact with the same data at different
levels of abstraction." In particular, the Morphic interface allows users to inadvertently break their tools. Real tools protect against this - things are held together (say, by screws) and require special (meta) tools to disassemble. That experience has informed the design of the Newspeak UI. Inspection/manipulation of UI objects is always possible, but requires explicit action.

b. "All user content, be it tools or data, should be manipulable across devices and
between users." Yes, and there is a lot to be done in that space, especially when one enables offline work and therefore needs to deal with merge conflicts. This also relates to principle 6. "Substrates should make edit and interaction history a first-class resource that can be
reasoned about, combined across substrates, and used generatively in users' work.".

c. "Users should be able to change or appropriate their software in the course of use, without having to adopt a programmer's mindset removed from their task domain." This sounds ideal, but historically has been all but impossible to achieve. I think AI will change this.

++++++++++ REVIEW 3 (Jonathan Edwards) +++++++++
Nice summary of Clemens et al research program on Webstrates. He states a set of principles guiding his past and future work that would be useful to discuss at the workshop. Repeating them here for my own convenience:

1. There should be no technical distinction between manipulating your tools and manipulating your data.
2. All user content, be it tools or data, should be manipulable across devices and between users.
3. Substrates should support composability through transclusion, both for interaction in use and for software construction.
4. Users should be able to change or appropriate their software in the course of use, without having to adopt a programmer's mindset removed from their task domain.
5. Substrates are layered, with each layer exposing different semantics and affordances, enabling tools to interact with the same data at different levels of abstraction.
6. Substrates should make edit and interaction history a first-class resource that can be reasoned about, combined across substrates, and used generatively in users' work.

I agree with all of this, and just might want to add to principle 5 that there should be a base layer providing a common data model. The browser DOM has shown us the power of this architecture, and Webstrates itself has built upon that, and now underneath with Automerge.

++++++++++ REVIEW 4 (Camille Gobert) +++++++++
This statement provides a certain view of substrates informed by the author's own historical perspective and the development of a number of systems, including Webstrates and those built upon it, which provide a collaborative

software substrate that runs within a webpage. It presents six principles that substrates must or should exhibit according to the author, which can be used to both analyse existing substrates and design new ones.

I really appreciate the historical introduction and the relation with the instrumental interaction theory. Since Michel Beaudouin-Lafon and Wendy Mackay recently revisited and formalised their approach to (interaction) substrates in a paper recently published at CHI (cited as [12]; cf. footnote 1), I would be curious in hearing the author's opinion on how the relate—or not—to their own vision. In particular, the notion of "constraint" is central in Beaudouin-Lafon and Mackay's vision, more so than, e.g., computation (though the latter is required to enforce the former, of course). It seems to me that Webstrates combines these two notions: for example, DOM replica are kept in sync (constraint) but each peer's JS engine can run independent computation to process data found in the DOM.

The notion of layer also sparks some thoughts in my mind. While other participants argue in favour of bundling code, data and interaction into a single substrate, layers appear to challenge this assumption: each layer might store/compute extra information that is not present in other layers (or only under a different form), and each layer might offer different interaction opportunities. Does it mean that each layer is its own substrate? Or exposes different information that could be read and written by one or more substrates: for example, different programs might interaction with Automerge documents stored on the disk and different instruments might interact with DOM data loaded in a webstrate?

That being said, several principles strongly relate to submissions from other participants: for example, principle 3 (transclusion) reminds me of Gilad Bracha's statement while principle 6 (interaction history) echoes Jonathan Edwards' work.

++++++++++ REVIEW 5 (Yann Trividic) +++++++++
The vision statement draws a series of six principles for qualifying substrates, and grounds it into the authors past projects and experience with this concept. It anchors the work in a very coherent history and lineage of projects which explore the concept of substrate in different contexts.
This contribution answers well to some of what is advocated for in other contributions: it showcases actionable, usable, and powerful prototypes in an accessible fashion. For example, Webstrates runs in the browser and has been used in a variety of contexts.

Additionnally, the metaphors that are presented here are very clear and self-explaining. Anybody without prior computer science knowledge can understand the concept of substrate and instruments through them. It is, to me, one of the objectives we could aim for this workshop: formulate a clear explanation for what we believe a substrate should be.

On another note, I wonder if the six presented principles are considered by the author as axiomatic principles. From what I understand, this approach proposes an interesting contrast to the _possibilies_ Basman refers to in his vision paper;

(Same as at least one other reviewer, I prefer signing my reviews, as it is not clear why they should be anonymous in this context!)

++++++++++ REVIEW 6 (Antranig Basman) +++++++++
Hi Clemens - I really appreciated this vision statement and of course Webstrates has been an inspiring sibling project for so many years, and a great source of ideas and technologies. I was particularly touched to hear some of the story of your early conversations with Wendy, Michel and James which has been one of the most fruitful collaborations that I know of.

Similarly to my response to Jonathan's vision statement, I'd like to go through your 6 desirable properties of a substrate, each of which I strongly support, and report where I feel they lie within our "hierarchy of needs".

1. There should be no technical distinction between manipulating your tools and manipulating your data

This principle is definitional, and aligns well with Jonathan's principle #5 "Programming & using are on a spectrum, not distinct." and hence the Malleable Systems "Software must be as easy to change as it is to use it" and one could say is implied by these principles although I appreciate something fresh is being said here. The "tools/data" dichotomy is aligned with, though not perfectly mirroring the "programmer/user" dichotomy so one would expect that bringing the latter onto a continuum, the same would be done for the former.
However there is an important subtlety which should be brought out here - whilst it's true there should be no "technical distinction" between manipulating tools and data, it's important that nonetheless, at runtime, some kind of distinction can nonetheless be made. This echoes a point from your 2023 "Principle and Pragmatism" paper where you report in Webstrates, through lumping content of all kinds into the DOM, that "An important consideration in designing future tools might be to allow users to separate versions caused by changes in program code from versions caused by application use." - if these sources of manipulation can't be cleanly disentangled, the substrate history will become chaotic. This is a problem which Infusion 6's layer-based development solves - the substrate's state is decomposed into layers each of which is tagged with a "variety" (e.g. "framework", "live", etc.) field which enables the provenance of changes in substrate state to be tracked and filtered on.

2: All user content, be it tools or data, should be manipulable across devices and between users.

This is important, and perhaps comes more into relief if one considers what considerations might *prevent* user content being manipulable across devices and users. To me this is a statement about the "permeability" of substrate content - how easy is it to import and export parts of the substrate relevant to the user into easily intelligible, simply structured files (such as JSON, HTML, etc.) and how easy is it to assign addresses to these parts so that across devices/users it is easy to agree which part of the substrate one is talking about. Smalltalk, for example, places unnecessary barriers in this area as a result of its "image problem".

3: Substrates should support composability through transclusion, both for interaction in use and for software construction.

This is central, and is a problem definitively solved in Infusion 6. I've always felt that Ted Nelson's view of transclusion was deficient, especially as embodied in the traditional iframe-based transclusion where the transcluded content lives in an unrelated, insulated world from its surroundings - even essentially in its own VM. It also doesn't participate naturally in the browser's layout and interaction primitives. For flat, informational content as Ted envisioned this was acceptable but for living, interacting material - especially for something which enjoys some incarnation of "scope" creates serious problems. In Infusion 6 through its model of "live layers" it is not only possible to transclude an individual live component into either the background of a static, or else a surrounding which is live itself, but also possible to composite multiple such components from different sources onto the same transclusion site - "phenotropically", one could say.

4: Users should be able to change or appropriate their software in the course of
use, without having to adopt a programmer's mindset removed from their task domain.

I connect this with my notion of "disclosure" - that in the ordinary context of use, some affordance is available which brings the capabilities of the substrate into view, in a way that remains aligned with the original context.

5. Substrates are layered, with each layer exposing different semantics and affordances, enabling tools to interact with the same data at different levels of abstraction.

I agree. Pavel has asked in a review to my statement whether the kinds of layers you talk about here are the same as Infusion's layers, which I could answer here: Not quite. Infusion's layered design *could* be used to build this kind of layered revelation of different levels of abstraction, but primarily they are a composition device to be used at a single level of abstraction. They are the equivalent of what in object-oriented programming were base classes/derived classes, or in prototypal programming formed the prototype chain.

6: Substrates should make edit and interaction history a first-class resource that can be
reasoned about, combined across substrates, and used generatively in users' work.

I agree, and see a good connection here with point 2. The forces preventing substrate content being shared across users and devices are the same forces which prevent reuse across time: if substrate contents are not straightforwardly serialisable and addressible, versioning the system becomes unmanageable since it is unclear how to restore a historical piece of substrate content in a usable form in the present, without having to drag the entire past substrate content along with it. There's also a connection with point 1: if you have to use categorically different means to manipulate tools and data, you likely have to use categorically different means to version them. And the same solution is helpful: being able to tag substrate provenance as framework, user design, user data etc. makes it possible to create a coherent edit and interaction history, avoiding the problem that I refer to from your "Principle and Pragmatism" paper.

Looking forward to talking over these principles some more tomorrow and building them into an engaging history of our community's thought. Thanks again for being such lovely neighbours!