MyWebstrates: Webstrates as Local-first Software

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

James R. Eagan
LTCI, Télécom Paris
Institut Polytechnique de Paris
Palaiseau, France
james.eagan@telecom-paris.fr

Peter van Hardenberg
Ink & Switch
San Francisco, USA
pvh@inkandswitch.com

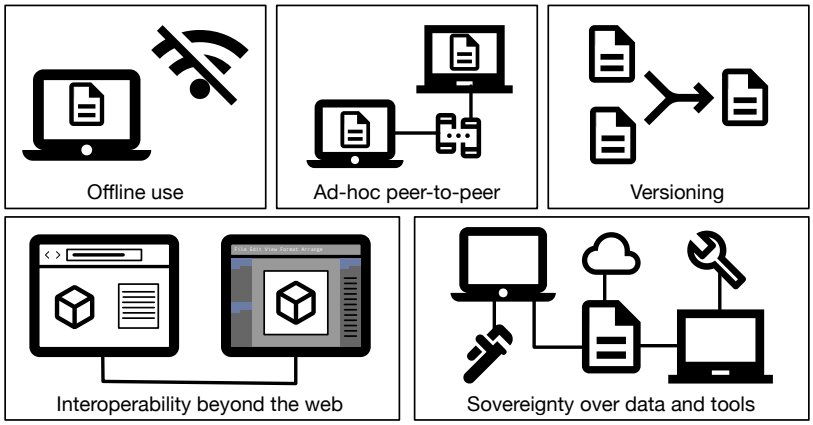

Figure 1: MyWebstrates is a local-first implementation of Webstrates enabling: offline use; ad-hoc peer-to-peer collaboration without an internet connection; versioning including clone and merge; interoperability beyond the web with conventional desktop software; personal and collective sovereignty over how data and tools are stored and shared.

ABSTRACT

Webstrates are web substrates, a practical realization of shareable dynamic media under which distributability, shareability, and malleability are fundamental software principles. Webstrates blur the distinction between application and document in a way that enables users to share, repurpose, and refit software across a variety of domains, but its reliance on a central server constrains its use; it is at odds with personal and collective control of data; and limits applications to the web. We extend the fundamental principles to include *interoperability* and *sovereignty* over data and propose MyWebstrates, an implementation of Webstrates on top of a new, lower-level substrate for synchronization built around local-first software principles. MyWebstrates registers itself in the user's browser and function as a piece of local software that can selectively synchronise data over sync servers or peer-to-peer connections. We show how MyWebstrates extends Webstrates to enable offline collaborative use, interoperate between Webstrates on non-web technologies such as Unity, and maintain personal and collective sovereignty over data. We demonstrate how this enables new types of applications of Webstrates and discuss limitations of this approach and new challenges that it reveals.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**.

KEYWORDS

Local-first software, malleable software, collaborative software

ACM Reference Format:
Clemens Nylandsted Klokmose, James R. Eagan, and Peter van Hardenberg. 2024. MyWebstrates: Webstrates as Local-first Software. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3654777.3676445

1 INTRODUCTION

Webstrates [28] presented a practical web-based realisation of *shareable dynamic media*; a vision of software inspired by early work of Kay and Goldberg [24], where software is malleable and shareable by its users, and distributable across their devices. Shareable dynamic media is defined as collections of *information substrates*: *"... software artifacts that embody content, computation and interaction, effectively blurring the distinction between documents and applications"* [28]. Users can share, repurpose, and recombine these substrates as they see fit. Webstrates or *web substrates* show how

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676445

Web technology can be leveraged for a practical implementation of shareable dynamic media by applying concurrency control using operational transformation [48] on the document object model (DOM) of a webpage, and applying the principle of transclusion for software composition. This demonstrates the technical realizability of the vision and has lead to numerous applications of the shareable dynamic media principles across different domains including computational and scientific notebooks [40, 43], video editing [29], video conferencing [17, 18], data visualisation [2], qualitative data analysis [33], teaching [38], and creative work [11, 13].

Webstrates represents an ambition to empower users to take control and ownership over their software by blurring the technical distinction between using and developing software. This echoes the ambitions of Smalltalk [25], Self [51], Lisp [47] and Hypercard [16], but Webstrates emphasises *shareability*; "*users can collaborate seamlessly on multiple types of data within a document, using their own personalized views and tools*" [28]. In Webstrates, software can even be adapted and reprogrammed collaboratively while in use to, e.g., let a user be assisted by a more capable peer in changing their user interface as they are using it [28, 40]. However, the original realisation of Webstrates was bounded by the technical constraints of the time, resulting in a centralised architecture where users are tied to a central server without means for working offline or for collaborating across servers. Webstrates allows users to work on the same document, each using their own tools, but only if they use the same server. In this way, the centralised server effectively "owns" and controls the user's data. This violates what we refer to as *personal and collective digital sovereignty* (inspired by Couture and Toupin [9]): the control of one's data and software tools and autonomy in choice of where and how to store them. Webstrates is also limited in interoperability beyond the Web and beyond a specific Webstrates server. While it does support some degree of interoperability through the open standards of the Web, it is largely trapped inside the browser making it difficult to interoperate with software outside of the Web ecosystem.

Webstrates builds its ideas of shareable dynamic media by applying two core concepts inspired by Beaudouin-Lafon: information substrates [6] and instrumental interaction [5]. Information substrates are layered, with each layer having its own organizing set of abstractions and affordances for interaction. Instrumental interaction in turn draws on the insight that interaction is frequently indirect, mediated by collections of instruments or tools, which are frequently polymorphic [7].

In Webstrates, the base substrate is the webstrate, which persists and synchronizes a web page's DOM, including any CSS and JavaScript. Tools built atop this layer compose in higher order information substrates, from general purpose tools that add code editing and execution for computational media, as in Codestrates [8]; to collaborative video editing, as with Videostrates [29]; or even substrates built around more specialized concepts as affinity diagramming and coding for qualitative data analysis [33]. These higher order substrates each stack and compose substrates to build off of each other, but the base layer substrate is the persisted and synchronized DOM.

With MyWebstrates, we introduce a layer *below* Webstrates' persisted DOM: *shareable data substrates*. The Webstrates paper articulated shareable dynamic media as having three key properties: *malleability*, *shareability*, and *distributability*. With MyWebstrates, we propose two additional fundamental properties: *interoperability* and *sovereignty* and show how they can be practically realized. As with Webstrates, the shareable data substrates provide persistence, synchronisation, and history support. They are, however, local-first, have branching versioning support, are platform independent, and are network agnostic.

Real-time collaborative editors have historically been notoriously difficult to build using a decentralised architecture, leading to most such systems relying on a centralised cloud server. This includes Webstrates as well as Google Docs, Overleaf (which we use for this paper), Figma, etc. *Local-first software* [27] is a software model presented as an alternative to the entrenched cloud model that applies recent results in conflict-free replicated data types (CRDTs) [45] for creating decentralised software where the users are in control of their data yet retain the means for collaboration as provided in conventional cloud-based software.

In this paper, we show how Webstrates can be redesigned as local-first software and demonstrate how this opens up for new use cases in terms of offline use, interoperability, and personal as well as collective sovereignty over data (see figure 1). Our examples serve both as pedagogical tools to explain how MyWebstrates works, as well as an evaluation of the contribution through their novelty [30]. MyWebstrates is open source[1], and a public installation is available at https://my.webstrates.net.

2 FROM WEBSTRATES TO MYWEBSTRATES

Webstrates [28] allows for creating collaborative and reprogrammable software through a conceptually simple change to how the web normally works. Conventionally, the document object model (DOM) of a webpage is an ephemeral representation of data fetched from a database. In Webstrates, a web-page is called a *webstrate*. The contents of its DOM are the data and changes to it are made persistent and synchronized to any other clients that may have the same webstrate open. By combining this mechanism with transclusion—realised through embedding webstrates using iFrames—Klokmose et al. [28] show how the traditional separation between applications and documents can be softened, and multiple users can collaborate on the same document with different tools. They demonstrate a document editor where two users collaborate on a paper with both functionally and aesthetically different editors. Also, that a more experienced user could remotely assist in changing the editor of another use while in use, to, e.g., add a new tool for managing references as is an example in the paper.

Since its publication, convenience APIs and constructs have been added to the Webstrates platform to allow it to scale to, e.g., realise complex software such as Vistrates [2] or Mirrorverse [18].

What enabled Webstrates, was applying operational transformation (OT) [48] to the DOM, so when multiple clients edited the same webstrate eventually it would reach a consistent state. This includes edits to JavaScript code embedded in the webstrate as well. Webstrates is based on a JavaScript implementation of OT called ShareJS [14] (now ShareDB [46]). OT is a first-generation technology for real-time collaborative editing and requires a central

[1]Released under the MIT licence at https://github.com/Webstrates/MyWebstrates/

authority to guarantee consistency. This leads to several limitations: A centralised architecture, no support for offline editing, and limited cross-server collaboration. Furthermore, the core synchronisation algorithm in ShareJS is implemented in JavaScript, making it difficult to interoperate beyond the Web.

While it is possible to run multiple Webstrates servers, for two users to collaborate, they must both have their documents and tools on the same server. It is not practically possible to work offline. To do that in a stable manner one would have to host a Webstrates-server on the local device, which is challenging on mobile devices. The centralised architecture also means that the user must entrust their data to the server owner.

MyWebstrates is Webstrates as local-first software and is compatible with existing software built for Webstrates. The transformative principle of MyWebstrates is that webstrates are created and reside first and foremost in the users' local browser instead of on a central server. Then, they can be selectively shared across sync servers or alternatively over a peer-to-peer connection. As such, MyWebstrates introduces lower-level substrates *below* the webstrates. It is effectively a drop-in replacement for the previous Webstrates server. But, where the data layer below the DOM in the original Webstrates was an opaque implementation detail, it is in MyWebstrates exposed as a layer of the substrate with its own capabilities and affordances; it can be used to interoperate beyond Webstrates and the Web as well as store arbitrary data without the impedance mismatch created when attempting to store data ill-fitted to the DOM in the DOM (as illustrated in section 4.4).

3 RELATED WORK

3.1 Local-first software

The term *local-first software* was introduced by Kleppmann et al. [27] to demonstrate an alternative to centralised cloud-based software while retaining the affordances of modern software that we have become accustomed to, such as real-time collaborative editing.

Kleppmann et al. [27]'s manifesto introduces seven ideals for local-first software: 1) *No-spinners:* all data operations are handled on the local device and synchronization to other devices happens in the background; 2) *Your work is not trapped on one device:* it should be possible to seamlessly distribute work across multiple devices; 3) *The network is optional:* software should work fully offline or with an unreliable network; 4) *Seamless collaboration with your colleagues:* rich collaboration should be possible across apps and file formats with powerful mechanisms for resolving conflicts; 5) *The long now:* data and software should withstand a cloud-service shutting down or going out of business; 6) *Security and Privacy by Default:* sensitive data does not have to be stored on other people's computers, and if it does it can be encrypted; 7) *You Retain Ultimate Ownership and Control:* The user can do with their data as they want and can process it in arbitrary ways. MyWebstrates addresses points 1-4, 7, and partially 5 and 6. We discuss these further in section 6.7.

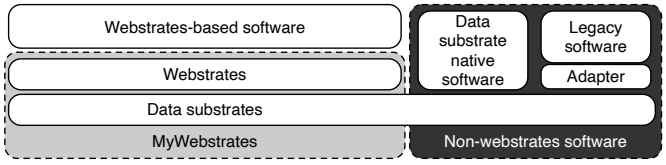

Figure 2: MyWebstrates introduces data substrates beneath Webstrates' regular DOM-based substrates. Existing Webstrates-based software can run on top of MyWebstrates. Data substrates can be used for interoperability with non-Webstrates-based software. This could be native software to the data-substrates—such as Ink & Switch's Tiny Essay Editor (section 4.4.2)—or legacy software with an adapter for the data substrates—e.g., the Godot game engine (section 4.4.3).

Kleppmann et al. [27] further propose using *conflict-free replicated datatypes* (CRDTs) [45] as a foundation for local-first software[2]. CRDTs provide an alternative approach to operational transforms for concurrent editing with robust peer-to-peer concurrency. The practical realisability of CRDT-based local-first software has been made possible through a number of open source implementations such as Kevin Jahn's Y.js [22] or Kleppmann and Ink & Switch's Automerge [26].

MyWebstrates is built using Automerge [26]. Automerge synchronizes changes to arbitrary JSON documents across clients connected in a peer-to-peer fashion. It provides rich versioning including conflict resolution, history navigation, as well as branching and merging of JSON documents. Automerge-Repo is a framework on top of Automerge to handle network connection and data storage. It is network and storage agnostic, and allows developers to create adaptors for new types of network or storage. The core of Automerge is written in Rust and is portable across platforms. To run in the browser, the core is compiled to web assembly (wasm).

Web-based local-first software has additionally been made possible by new standardised browser features such as service workers [37], which enable offline loading and handling of HTTP requests. Storing large amounts of data locally is possible through the IndexedDB API [36].

Commercial platforms for developing web-based local-first software includes DxOS [42] and Replicache [44] among others.

3.2 Interoperable systems

Historically, interoperability between programs happened through the file system and standardized file types. In the UNIX philosophy, interoperability is central, and common practice is to achieve complex tasks by piping data between simple programs.

Proprietary file formats and later software as a service make interoperability between software increasingly difficult, trapping users in so-called application silos.

Ubiquitous computing and computing beyond conventional personal computers created a push towards interoperability between applications running on heterogeneous devices in dynamic networks. In research, for example, the recombinant computing project [12] let devices exchange capabilities over an ad-hoc network and interact using meta interfaces agreed upon at runtime. PatchPanel [3] uses the iROS event heap to enable interaction between heterogeneous software. Shared Substance [15] proposed a data-oriented approach to creating software that spanned multiple devices by

[2]CRDTs were also mentioned as a future avenue for overcoming implementation limitations in the original Webstrates paper [28]

decoupling shared data from behavior; as contrasted with conventional remote method invocation or (RMI) or the common object request broker architecture (CORBA). Here, a cross-device application was represented in a shared graph structure similar to a scene graph and software running on the different peers could dynamically associate device-appropriate behaviour to the data. Our data substrates share similarities with Shared Substance, although all shared data in MyWebstrates is replicated between peers.

3.3 Related philosophies

MyWebstrates is inspired by various related philosophies around the shape of software, notions of digital sovereignty, malleability and appropriation, and expressivity.

One of the main related techno-philosophical efforts around decentralisation is the *fediverse*: heterogenous web-based softwares that interoperate over the W3C ActivityPub protocol [54] for decentralized social media. With ActivityPub, services publish different kinds of content streams related to various actors through a flexible, standardized protocol. Users subscribe to these social activity streams using the tools of their choice, which interoperate through ActivityPub. Thus, a user could subscribe to a (Twitter-like) stream combining microblog posts, photos shared through a Flickr-like photo-sharing service such as Pixelfed [49], or a Reddit-style meta-community like Kbin [53]). The protocol facilitates interoperability, with users freely mixing and merging content across these streams, adapted to the modality of the consuming tool. Similarly, tools such as Matrix [34] focus on decentralised instant messages where chat rooms are federated across servers or even bridged between different services such as WhatsApp, Mattermost, and Slack. In all of these approaches, users freely mix content across servers, services, and media, as facilitated by an underlying, federated protocol.

Other work aims to explore new architectures for hypermedia. For example, the HyperHyperSpace project [4] is perhaps most similar in approach to MyWebstrates and aims to provide "a local data store, both in-browser using IndexedDB and server-side; a data representation format, based on Merkle-DAGs and CRDTs; and a secure data sync protocol over WebRTC and WebSockets."

On the other end of the spectrum, the Files over Apps philosophy takes a different approach, focusing on data in files and formats that are easy to retrieve and read, under user control [1]. These explorations further probe a philosophical tension between data-first software, open-source, and proprietary software formats, and the complexity of media ownership and stewardship. MyWebstrates extends the Webstrates vision of shareability and malleability to include interoperability and sovereignty, while aiming to provide a collection of fairly simple but powerful substrates in which shareable dynamic media can grow.

4 BASIC USE AND USE CASES

In the following section we present the basic usage of MyWebstrates as well as demonstrate new use cases that it enables. The use cases are illustrated in the accompanying video.

4.1 Basic use

Alice creates a new Webstrate to capture her notes. Since she has never used MyWebstrates before, she first visits a website that hosts the client, such as `my.webstrates.net`. This static website installs the MyWebstrates client in her browser. She then navigates to `/new`, e.g., `my.webstrates.net/new`, to create and load a new blank webstrate (or, with optional parameters, say, a new codestrate with a built-in editor [8]). She now writes her notes in the webstrate, such as by using the browser's developer tools, content-editable, or codestrate's editor, depending on how she chose to create the new webstrate. She then closes her browser and takes a break. When she returns, she re-opens the webstrate URL, which shows her notes as she had entered them.

Alice decides she would like to share her notes with Bjørn, so she registers her webstrate with a sync server. She can either do it manually in the console with `webstrate.addSyncServer('sync.webstrates.net')` or through a menu item in codestrates. To share the webstrate with Bjørn, she appends `@sync.webstrates.net` the the URL; Codestrates has a button for this. In contrast to Webstrates, MyWebstrates does not share webstrates by default. When Bjørn fetches this URL, the MyWebstrates client installed in his browser will fetch and synchronize through the specified sync server. He adds his comments, which are live-synchronized with Alice. If Bjørn makes edits that Alice doesn't agree with, she can roll back the changes to a previous version.

4.2 Personal and collective digital sovereignty

Anne is preparing to teach an informatics class using the Webstrates-based CoTinker platform [38]. CoTinker helps create Webstrates-based learning activities for high-school students involving collaborative programming. Anne wants to be sure to protect the privacy (and legal compliance) of her students' data and logs of their activity, which webstrates need to capture, synchronize, and store. The school has an agreement with a GDPR-compliant cloud provider where Anne can spin up a dedicated sync server only for the students in her class to use. Anne syncs the prototype of her learning activity to the server for the students to copy. At the end of the term, she spins down the server, deleting its stored data. Students retain their own local copies.

As such, MyWebstrates allows user control over where their data and software are stored. When creating a webstrate, it is first and foremost created locally on the user's device and does not leave the device unless explicitly shared. Sharing is typically done by federating a webstrate to a sync server. Users may have private sync servers to synchronise their webstrates between devices and for backup purposes. Such a private sync server could be hosted on a home NAS or through a rented cloud server. A sync server can be personal or collectively owned, e.g., by a household or a company.

During class, Anne realises that one step of the assignment needs further explanation. She has a local slide editing webstrate that can transclude the slideshow provided to students. Using this, she adds a slide with explanation to the prototype of the learning activity on the shared sync server. Afterwards, she asks her students to press the update button in their copies to pull in the changes.

The original Webstrates paper [28] as well as the Videostrates paper [29] demonstrate scenarios where users collaborate on a shared webstrate with different tools—so called *asymmetric collaboration*—using transclusion. With MyWebstrates, these tools and documents do not have to reside on the same server. Personal tools

may reside locally or on a private sync server while the shared document can reside on a publicly accessible sync server or the other way around. In this way, users can exercise more granular control over the storage and synchronization of their data.

4.3 Offline use

MyWebstrates enables offline use in the following forms: single-user offline work, intermittent offline work, local peer-to-peer work, and clone and merge.

4.3.1 Single-user offline work. The prototypical offline use-cases are working on an airplane, or if one of Anne's students takes their laptop outside in the school yard to work. Historically, this kind of offline use is the most common context of use for a personal computer, but it has been made challenging with software-as-a-service and cloud-based software. In MyWebstrates, single-user offline work without a connection to a server is possible 1) once the client has been installed in the user's browser by having previously navigated to its page, and 2) as long as the webstrates being worked on have been previously loaded—placing them in local browser storage—or were created during the offline session. The implementation of a given webstrate must accommodate offline use by, e.g., not relying on external asynchronous data requests or not having dependencies on online code that cannot be cached.

4.3.2 Intermittent offline work. Intermittent offline work happens when two or more users are collaborating synchronously and one of them intermittently loses connectivity, e.g., when entering a tunnel on a train or due to an unstable or overloaded school network.

As MyWebstrates is local-first, changes to a webstrate are made locally and synced to peers if available. If a connection is lost and reestablished, changes from other peers will be automatically merged. Here, conflicts can potentially arise (see section 6.2).

4.3.3 Local peer-to-peer work. Users may wish to collaborate without access to the internet. For example, Alice and Bjørn might wish to work on a slideshow on an airplane, or Anne's students might edit science reports together while outdoors on a field trip. Traditional cloud software requires access to centralized servers to enable this, but because MyWebstrates users have a full copy of the software and data locally, collaboration can occur via a peer-to-peer connection over an ad-hoc network. For example, to establish a peer-to-peer connection between two browsers, Alice and Bjørn can each navigate to `/p2p`, which provides an interface to establish a peer-to-peer connection over an ad-hoc WiFi network using a QR-code based offer and response exchange. This ad-hoc connection will then remain active until one of them closes their `/p2p` tab. While this peer-to-peer connection is established, Alice and Bjørn can share any webstrate directly between their browsers.

4.3.4 Clone and merge. One of Anne's students, Céline, is presenting the results of their group work to the class. While she is speaking, her group mate Lee realises that their solution to the last exercise has a bug. Lee quickly clones the learning activity, fixes the bug and tests that is is correct, then merges the changes into the main document before Céline reaches the final exercise.

Clone and merge can be useful in such situations where a user wants to edit a shared webstrate in an offline copy and merge the changes back into the shared one. This is a common situation when developing software that is in use by others, or when for any reason not wanting to share intermediate changes before they are ready. MyWebstrates lets a user make a clone of a webstrate and merge changes back into the source webstrate. Merges are handled automatically, however conflicts can occur (see discussion in section 6.2). There are two ways to duplicate a webstrate: with `webstrate.copy()`, which creates and opens a new copy without version history, and `webstrate.clone()`, which creates a webstrate that retains the version history of the original and also allows for calling `webstrate.merge()` on the original to merge in any changes. All of this functionality can be accessed through the console or through more convenient UI built into higher-level webstrates, e.g., through Codestrates.

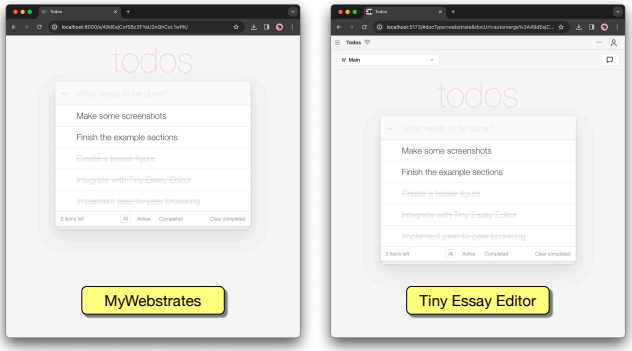

Figure 3: A simple ToDo-list implemented as a webstrate (left) and loaded in the Tiny Essay Editor through the data substrate (right).

4.4 Interoperability

MyWebstrates introduces new opportunities for interoperability in terms of users being able to apply different tools to simultaneously edit data in the same webstrate. We identify three cases of interoperability: *webstrates in foreign software*, *foreign data in webstrates*, and *data substrates-based interoperability*.

4.4.1 Webstrates in foreign software. Bjørn and Amélie are collaborating on a writing project, and Bjørn has created a ToDo list webstrate to coordinate their work. Amélie is used to working in her own text editor, and opens the ToDo webstrate from within. She wants to privately brainstorm on a list of tasks and creates a branch of the ToDo-list using the editor's built in versioning UI. When satisfied, she merges the changes back into the original.

As a webstrate is stored in a general data substrate realised through Automerge, any software capable of communicating with an Automerge-based sync server can load a webstrate. This means that the tool for navigating the history of an Automerge document as developed by Ink & Switch [20] can be applied to a webstrate simply by sharing it over a commonly accessible sync server.

Ink & Switch has developed an Automerge-based open source editor for writing essays called Tiny Essay Editor (TEE) [21]. TEE can be extended with new data types, and Webstrates' DOM model can be treated as a data type. As an experiment, we transplanted

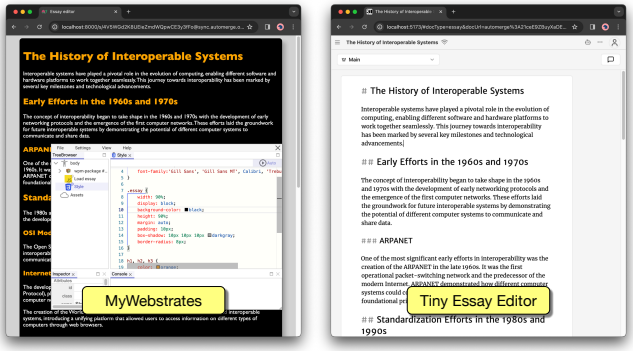

Figure 4: A Tiny Essay Editor essay (right) loaded in a Codestrates-based webstrate environment (left) editing custom CSS through the Cauldron editor.

the core of the Webstrates client wholesale into TEE and could create and open webstrates. Changes synchronize bidirectionally with MyWebstrates through the data substrate (Figure 3). Moreover, branching and merging on a document in the TEE interface is seamlessly applied to the webstrate.

4.4.2 Foreign data in webstrates. Bjørn has a Webstrates-based tool for applying expressive graphical styling to documents. Amélie has written some text in Ink & Switch's Tiny Essay Editor that will be printed using Bjørn's tool. He transcludes Amélie's essay into his Codestrates-based styling editor. While he tweaks the custom styling in the Cauldron IDE (Figure 4), Amélie's edits continue to synchronize live—with his new styling displayed in his interface.

In MyWebstrates, a webstrate provides direct access to the data substrate and its Automerge instance. Connections can be made programmatically to any Automerge sync server, and any Automerge document can be loaded and made accessible to code in a webstrate.

4.4.3 Data substrates-based interoperability. Anne and Dominique are working on level design for a web-based online game for a customer. The level is represented in the universal scene description (USD) format [41], which Anne has been editing through the Godot game engine while Dominique has been using his preferred environment, Unity. In an online meeting with their client, they share a link to a web-based representation of the level and can do live edits based on the feedback from the client during the meeting.

It is possible to store arbitrary data in the data substrate layer of a webstrate and programmatically listen to changes to it. We have experimented with storing a scenegraph of a 3D environment formatted as a JSON representation of the universal scene description (USD) format [41]. We implemented support for loading an Automerge-based webstrate in both Godot and Unity game engines over a sync server with live updating of the scene when remote changes were made to the webstrate.

Figure 5 (left) shows a scene loaded in Godot, Unity, and a web browser with ThreeJS. The scene is edited through Codestrates' Cauldron IDE, and changes live-update in all views.

This approach can also enable separation of interaction logic and business logic in distributed applications. In figure 5 (right), a Tetris level is expressed in USD and rendered in 3D in Godot. A Java application shows a 2D view of the scene and allows for manipulating the model with the mouse, e.g., to rearrange bits of a Tetris piece. The game logic is running as JavaScript in a Web browser (not shown), that also takes keyboard input (potentially from multiple clients) to control the pieces. The Java application is not aware of the ongoing Tetris game and just knows how to render and manipulate USD cubes in a 2D plane. This means that it is possible to take the level apart through the Java application while the game is running—effectively performing instrumental interaction [5] on a USD scene (see accompanying video).

5 IMPLEMENTATION

MyWebstrates consists of four core components: a client, a service worker, optional synchronization servers, and a peer broker mechanism for multi-user offline work.

All webstrates are represented as JSON objects persisted in Automerge documents. Each webstrate document has four core components: `meta`, `assets`, `data`, and `dom`. `meta` can include information such as creation time or id of the webstrate it has been copied from. It also contains the list of zero or more sync servers that the webstrate is federated to. `assets` holds references to any file-based assets that may be stored in the webstrate (e.g., binary images or JavaScript libraries). `data` can hold arbitrary data, e.g., a USD-based scenegraph, as shown in section 4.4.3. `dom` holds the document object model of the webstrate stored as JsonML [23]. A webstrate is identified with a universally unique identifier (UUID) that is part of its URL. The id is globally unique, as a shared webstrate is federated between clients and sync servers with no one version being the authoritative version[3].

5.1 JavaScript Client

The MyWebstrates client is responsible for maintaining correspondence between the state of the DOM in a webstrate and the underlying Automerge documents that stores it. This client can be served as plain files from any static web host or even by temporarily running `python -m http.server` in the client folder. Unlike the original Webstrates, no server-side code is required to operate.

The client includes a service worker [37] which caches every part of the system to ensure continued use without an internet connection. It has other responsibilities described below.

The client combines two main elements: the Webstrates DOM synchronization system and the Automerge document synchronization system. Changes to the DOM are translated into changes to an Automerge CRDT document. Automerge records those changes and stores them locally (in IndexedDB) as well as synchronizing them to any other online collaborators (as described below.) Other browser tabs are synchronized using the BroadcastChannel feature of the browser without any network access required.

To determine which webstrate the user is working with, the id of the webstrate is extracted from the browser's current URL (e.g., `/s/b7VabMk9pCKkg/`) and the corresponding webstrate document is loaded from Automerge.

[3]We expect that a future version will provide a scheme for nicer names.

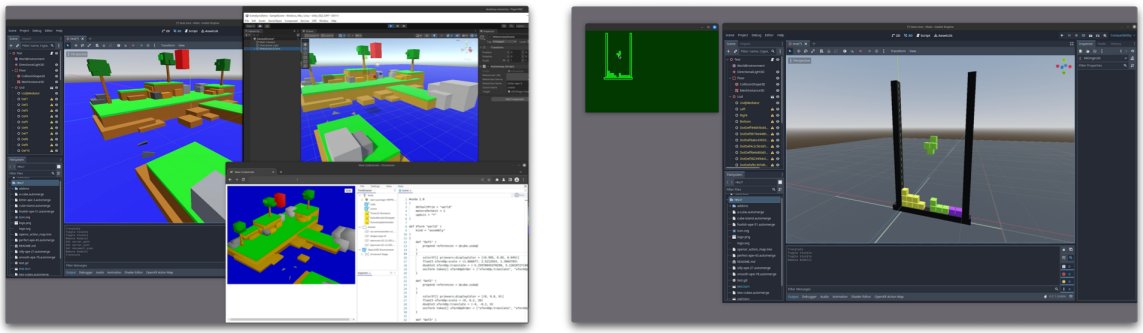

Figure 5: *Left*: A 3D scene stored in USD represented as JSON in a webstrate rendered in Godot, Unity and a Web browser using ThreeJS. Edits done in the webstrate are synchronized to all visualisations. *Right*: A Tetris game represented in USD rendered in Godot and in 2D in a custom Java application. The game logic and control is handled off-screen in a webstrate running in a Web browser. The Java application can manipulate the pieces while the game is ongoing.

Because a webstrate can be stored on any Automerge sync server, either the document itself or the URL it is loaded from can recommend a hostname for any sync servers where the data is available. This allows users to operate their own sovereign hosts but still to collaborate seamlessly. This is done by appending `@sync.some-server .net` to the URL. At runtime, the client will establish a new connection to the referenced sync server and request the data.

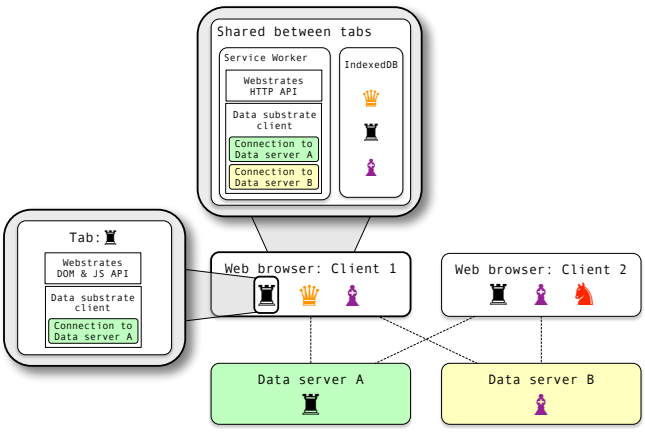

Figure 6: The bottom right side of the figure shows two browser clients that each have three webstrates open; each webstrate represented by a colored chess piece. The black tower and the purple bishop are shared between the two clients. Black tower through sync server A and purple bishop through sync server B. Yellow king and red knight are not federated to any sync servers. The left side of the figure magnifies the internals of the tab with the black tower in client 1. Each tab runs an instance of the Webstrates client, an Automerge instance setup with any socket connections to sync servers the given webstrate might be federated to. The top shows what is shared between tabs; a service worker and the IndexedDB data store. The service worker has its own Automerge instance with socket connections to federated sync servers of all open webstrates. The IndexedDB holds all open (and previously opened) webstrates of the client.

The client then parses the document's stored `dom` and populates the DOM in the browser. A two-way mapping is now created between the DOM in the browser and the automerge document using the same MutationObserver-based strategy as in the orignal Webstrates implementations [28] and the patch API of Automerge[4].

Ephemeral messaging is implemented using broadcasting over the sync servers. This broadcasting is similarly used when establishing WebRTC based connections between clients of the same webstrate for, e.g., video communication or screen sharing; in Webstrates terminology, this is called signal streaming.[5]

The client has a simple API for copying a webstrate (without history and with or without federations) or cloning a webstrate (with history) where changes can be merged into the original. It also implements a basic API for handling versioning, where it is possible to revert back to a specific version.

The client provides an API for adding assets to a webstrate through a call to `webstrate.uploadAsset()`. This could, e.g., be a binary image or any other type of file. Assets are stored in independent Automerge documents that are treated as immutable data. This way, updating an asset to a new file will preserve version history as an older version of the webstrate will just point to the previous asset document. Assets are served through the service worker (see below).

Automerge is exposed to the webstrate, which means that software running in a webstrate can access the data substrate directly on the `automerge` object. Automerge automatically resolves editing conflicts at the syntactic level of the document. A webstrate is effectively a set of nested maps and arrays. Edits are targeted to specific keys or insertion points: two users can concurrently edit a DOM element by updating attributes or insert children with no conflicts. If, e.g., two users replace the same attribute, it may cause a conflict, but Automerge selects a consistent winner. A history including conflicts is retained, and accessible through the exposed

[4]https://automerge.org/automerge/api-docs/js/types/next.PatchCallback.html
[5]https://webstrates.github.io/userguide/api/signaling.html

Automerge API. The MyWebstrates client ensures that no edits violate the structure of the DOM (e.g., adding multiple lists of children to an element). Some edits are harder to merge than others: for example, support for moving subtrees in Automerge is preliminary. Da and Kleppmann [10] propose an algorithm to improve handling this operation, but it is not yet available for use in Automerge. It would reduce a common class of semantic conflicts.

As such, MyWebstrates ensures that synchronization conflicts cannot leave a document in a syntactically invalid state, and maintains access to the full history of changes, including conflicting changes. Semantic conflicts, however, can still arise (see further discussion of conflicts in section 6.2).

5.2 Service worker

The service worker has the following responsibilities: 1) Serving client files from its cache, 2) serving assets, and 3) providing an HTTP wrapper for certain Webstrates API calls.

The service worker stores all client files. Once initially loaded, the client is accessible in the browser without a connection.

When adding an asset to a webstrate, that asset becomes available through a resource on the URL (e.g., `/s/2L...ct/logo.png`). The service worker will intercept the request for `logo.png`, lookup the asset document in the list of assets in the webstrate, load the asset doc from Automerge and respond with the data of the asset based on a stored MIME type. The service worker will also serve files from paths into zip-file assets.

The service worker furthermore provides a wrapper for some of the Webstrates client API to, e.g., allow creating new webstrates.

5.3 Sync server

MyWebstrates does not require a network connection to operate and can synchronize directly (as described below) with other clients. However, it is helpful to store data on a cloud server for durability and ease of replication. Otherwise, when a client is closed, its data is unavailable to anyone else, and if the computer should be lost, the data would be lost as well.

MyWebstrates uses a conventional Automerge sync server[6] as a store-and-forward host for collaborators. Any client can post edits to a webstrate if they know its id and hostname of the server.

Because this server is not specific to webstrates, it allows for interoperability with any other automerge-based applications that share it. The sync server is a simple service that accepts WebSocket connections and that stores documents in the file system.

5.4 Local peer-to-peer connectivity

For multi-user offline work, MyWebstrates provides a mechanism for establishing peer-to-peer connections over an ad-hoc network. This is possible because the underlying data is all stored locally and can be synchronized through the same mechanisms as updating a cloud-hosted sync server. Connections are established over local ad-hoc wifi networks via WebRTC and peer signalling occurs using QR codes that encode the standard WebRTC offer and response protocol. Each user opens a built-in page within the application. They face their laptops together and the camera on each reads QR codes presented by the other until the network connection is established (see figure 7). Once the network connection is established, the same protocol used by the cloud servers enables collaboration.

[6]https://github.com/automerge/automerge-repo-sync-server

6 DISCUSSION

6.1 History and versioning

Both Webstrates and MyWebstrates use a history of changes to maintain consistency, and it is possible to roll back to a previous version by applying an inverted list of changes.

In MyWebstrates, there is not a central server with an authoritative order of these changes. Hence, versions are no longer guaranteed incremental (..., v12, v13, ... v1701, etc). That is, there is no common mapping to order these change sets. As such, MyWebstrates uses Automerge's git-like hash of changes to globally identify a document's current version (e.g., 19624...f516c). For a given client (such as a browser or sync server), however, an internally-consistent incremental version number is available, but cannot be used to identify a given version across instances as each instance may have applied its patches in a different order. These incremental version numbers are more usable but are only locally valid, while version hashes are globally valid.

It is thus possible to identify a given version of a document (locally through the version number or globally through a version hash) and to roll back to previous versions, as with Webstrates, but the order of these versions is not globally stable.

In contrast to the original Webstrates, MyWebstrates can maintain history when cloning a document (similar to creating a fork in git). Any document with a common history can thus merge any subsequent changes. This makes it possible to create branches of documents that can pull in changes from other common ancestors.

6.2 Conflicts

Conflicts can be characterised along two dimensions: semantic vs. syntactic, and synchronous vs. asyncronous. Syntactic conflicts pertain to the document data structure while semantic conflicts pertain to the meaning users ascribe to the document. Both synchronous and asynchronous syntactic conflicts are handled automatically by Automerge (as described in section 5). Synchronous semantic

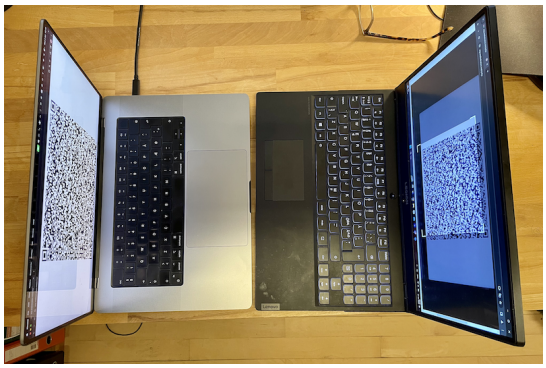

Figure 7: Two laptops on the same ad-hoc network face each other. Each presents QR codes detected by the camera of the other to execute the WebRTC offer and response protocol until a connection is established.

conflicts are localised in time and space, e.g., when two users are simultaneously editing the same sentence. These will often be detected and immediately remedied by users, typically through social negotiation. Asynchronous semantic conflicts on the other hand may happen unexpectedly and in distant parts of a document. Discovering and resolving these can be hard and requires tools to maintain awareness of changes and to inspect the history in order to remedy unintentional merges.

MyWebstrates provides low-level APIs to a webstrate's data substrate, from which one could build higher-level conflict managers. We deemed developing such tools out of scope of this paper. First, they are heavily application dependent: tools for conflict resolution in a MyWebstrates-based vector graphics editor would be significantly different from for a text editor. Second, conflict handling in CRDTs is a problem that is under active research and new results are currently being published on a regular basis (e.g., [10]).

Although different applications will have varied requirements for managing change review and conflict resolution, there may be opportunities for shared tooling. The Patchwork project [20] will allow for creating bespoke versioning tools for Automerge-based applications. They illustrate a versioning tool for a tldraw-based whiteboard [50] running on Automerge. With Patchwork, we would be able to create a generic history tool embedded in the Cauldron IDE that operates at the DOM level. Application developers could create domain-specific tools, e.g., for Mirrorverse or CoTinker.

We have not found that the current lack of support for advanced conflict resolution has prevented us from experimenting with building usable local-first software on top of MyWebstrates.

6.3 Personal and Collective Sovereignty

Webstrates aims to provide software users with the ability to adapt their tools to their own idiosyncratic needs, such as through bringing in their own personal tools and representations to collaborative documents. From this perspective, Webstrates has been fairly successful. Due to its centralised architecture, however, the locus of control remains in a cloud-based server.

MyWebstrates aims to bring data under a user's own control. Through its local-first architecture, data remains on the user's own device by default (in the browser's indexedDB). As such, the user maintains autonomy over any data and tools they create and use. If a user wishes to create, say, a personal journal, their intimate thoughts are not implicitly shared with another entity.

Collaboration, however, can be enabled in an opt-in fashion by registering a webstrate with a sync server. This explicit operation promotes data from being under an individual's locus of control to one that is shared between the user and others through the sync server. Moreover, collaboration often occurs across individual and collective boundaries. For example, this paper is a collaboration across three institutions, each of which has a legitimate motivation for control and ownership of its product. When collaborating through a traditional cloud-based tool (such as Overleaf or Webstrates), ownership and control of the data is effectively conferred to the cloud owner. Even with most federated tools such as Git, collaborators must agree upon a single controller to effectively "own" their data (such as a neutral third-party (e.g., GitLab) or one institution's locally-hosted Git Forge (e.g. GitLab, Forgejo, etc)). While it is possible to synchronize data across multiple Git servers, collaborators must elect one such repository to be canonical.

In MyWebstrates, we aim to provide a different approach. When shared with a sync server, each sync server effectively caches its own copy of the collective document. A document that lives in Alice's browser and Bob and Charlie's sync servers does not live in any one chosen place but rather in all. This paper could thus be collectively owned and controlled by all three partner sync servers.

This mechanism relies on documents having a global shared name, independent of the sync server or sync servers used for collaboration. In this way, ownership of the data means having a snapshot of the collective data under the user's (or collective's) locus of control. By making the data substrate's id globally unique, independent of which sync servers may be involved, the document is effectively independent of any specific actor. All participating sync servers may effectively hold the data, but no specific one maintains sole control or ownership.

The federated ownership of documents has the downside that once a document first been shared, it is extremely difficult to delete as it will require asking all peers to do so—peers that may be offline or potentially unwilling to fulfill the request. This is a fundamental problem with federated architectures including the ActivityPub-based Fediverse and email as well.

6.4 Interoperability

The concept of sovereignty extends beyond simply where data are stored. It includes notions of empowerment beyond the confines of application and document, including beyond Webstrates itself. Section 4.4 illustrates three such use cases: using Webstrates in Ink & Switch's Tiny Essay Editor (TEE); using TEE in a Webstrate; and combining conceptually divergent data between Webstrates and a game engine, such as Unity or Godot.

The first two examples show how a webstrate can be manipulated in a foreign software environment and vice versa, with each environment acting as its own set of collective instruments on a shared data substrate. While neither of these systems were designed with the other in mind, the user can freely import from the other. In this way, TEE and Webstrates could be thought of as alternative editors for a shared collaborative document, similar to the way that the original Webstrates [28] enabled Alice, Bob, and Charlie to use their own specialized tools in idiosyncratic ways to collaborate on a shared document. This interoperability is facilitated by the conceptual alignment between Webstrates' model of the DOM and TEE's model of an essay.

The Tetris example shows a more heterogenous sort of interoperability beyond the web, with parts of the logic written in Java, parts in Webstrates, and game models based on a USD scene graph embedded in Godot or Unity. Each of these pieces comes together, with live synchronization coming through the underlying data substrate. While some data can be represented directly through Webstrates, scene graphs and models are exposed through a webstrate's `data` property while still maintaining the underlying synchronisation and sharing model of the data substrate. This enables divergent software to co-exist and collaborate, similarly to the way that Substance Grise [15] grafted a shared scene graph into the Anatomist application used by neuroscientists. Whereas the former did so

through a bespoke plugin architecture, this software demonstration shows how a sharing-oriented data substrate can be generalized to more diverse applications, from 3D virtual worlds in Unity & Godot, to specialized tools with game physics and game controls shared across multiple actors.

One of the key challenges for this kind of interoperability is that of conceptual alignment between the underlying data formats. MyWebstrates' data substrates enable synchronisation at the raw data layer by using general abstractions for offline synchronisation. By building off of Automerge, any system that can embed this library can theoretically interoperate with MyWebstrates at this layer. The second challenge is being able to interpret and make sense of the exchanged data. For Webstrates and TEE, this is relatively straightforward since both use a JSON tree to describe their content. For the games example, these use more opaque data structures. Approaches such as Cambria's data lenses [31, 32] show promise at enabling translation between conceptually compatible formats, but may require more bespoke integration to interoperate.

6.5 Transclusion revisited

Transclusion is the referencing and embedding of other documents, in whole or in part [39]. In Webstrates, it is expressed using iFrames and is a key mechanism for composition and malleability. By transcluding a webstrate in another, the embedding webstrate can change the embedded webstrate's appearance and behavior. For example, Liu and Eagan [33] transclude interview transcripts in a textual editor for qualitative coding, while another webstrate transcludes those codes for an affinity-diagramming based representation on a wall-sized display, enabling asymmetric collaboration across devices, representations, interactions, and analytic processes.

MyWebstrates' data substrates enable a similar sort of transclusion across interoperable systems. In the Webstrates+TEE examples above, a user can transclude a webstrate into TEE (or vice versa), enabling her to combine distinct tools and environments to meet her own idiosyncratic needs. Or, collaborators could collectively work on the same document with their own personal tools[7].

In the Tetris example, transclusion is used between diverse systems and environments not to provide alternate lenses on the same data but to provide new specialized capabilities, with each part of the system extending the whole.

Transclusion in MyWebstrates is, as in the original Webstrates, limited to the granularity of a document; either a webstrate or data substrate-based document. Transcluding, e.g., a single slide from one slideshow to another requires either that the slide is itself a self-contained document or that the whole slideshow is transcluded and only the specific slide is rendered. Ideally, any node in a document would be independently addressable and transcludable, however, this would require a fundamentally different data model than the document-based model of Automerge.

6.6 Authentication and authorization

In the original Webstrates, an access control list is stored in the document and enforced by the server.

With MyWebstrates, the ID of a webstrate represents a basic security capability that gives lasting read and write access to the document to any user with access to the shared sync server. This is a pattern found in some other web applications, including today, the tldraw collaborative drawing system. Sharing a URL provides immediate collaborative access to a document. Users without the unguessable URL have no access to the document at all.

The Automerge system extends this security further by allowing users to provide their own sync servers: even the MyWebstrates development team has no way of knowing which sync servers exist.

The trade-off in this design is that today, there is no mechanism for revoking access to a document. Once the ID of a webstrate is known by another user, nothing short of forking the document to a new ID will allow excluding a collaborator.

This is particularly concerning in an environment where security boundaries between webstrates are weak and anyone can contribute code that everyone will execute in their browser. Clearly future work is required to develop a more robust model of secure collaborative code execution.

In simple cases, whether a document or a change should be shared with a peer could be handled through basic public/private key authentication. One challenge is group management, which could apply to either groups of people (such as a team of coauthors), or groups of devices (such as your laptop, phone, and desktop).

Consider, for example, a group with three users, Alice, Bob and Eve. Alice is a group administrator and removes Eve. Before Bob receives that message, he receives updates from Eve and takes actions based on it. Later Bob hears that Eve should have been removed earlier – how should Eve's updates be handled?

It gets even more challenging in the case of peers with shared administration. Consider the same group but where Alice and Eve are both administrators. Each removes the other from the group concurrently. Who should Bob consider a member of the group – and what should happen if Bob receives messages out of order?

While no solution is currently available for Automerge, Herb Caudill [19] sketches out a possible approach. Distributed key agreement [52] and Zelenka et al. [55]'s User-Controlled Authorization Network (UCAN) standard provide potential partial solutions.

6.7 Local-first principles

MyWebstrates addresses the seven principles of local-first software (introduced in Section 3.1) in the following manner: 1) *No spinners:* Document edits happen locally, and are not dependent on a network connection. As such, local changes happen more quickly during use. Initial load times to synchronise from a remote server, however, are potentially longer, bounded by the complexities of Automerge and available network connection. We expect this gradually to improve with optimizations to Automerge. 2) *Your work is not trapped on one device* is inherently supported. Moreover, MyWebstrates—in contrast to Webstrates—are not shared by default. The user must explicitly decide what to share, including with themselves across their own devices. Because the underlying data substrates are stored in the browser's IndexedDB, MyWebstrates are currently trapped in the specific browser or shared with a sync server. There is also an experimental Webstrates package that can write Codestrate fragments to the local file system using the web FileSystem APIs [35].

[7] such as when one co-author of this paper works in markdown+git and another in Overleaf

3) *The network is optional* is inherently supported by MyWebstrates, however, software developed using MyWebstrates could introduce network dependencies that limit offline use, as with a tool for remote conferencing that might depend on a specific conferencing server being available. 4) *Seamless collaboration with your colleagues* is inherently possible and is a fundamental goal of Webstrates. MyWebstates introduces an explicit step to activate sharing through a sync server, reflecting the tension between seamless collaboration and control over data. It also adds richer version control, including branching and merging, by delegating this responsibility to the lower level data substrate. 5) *The long now* is supported by users not having to rely on a centralised server, however, it also means that the responsibility for keeping data lies with the user to a larger degree than before. MyWebstrates may have a shorter long-term viability than simpler file storage formats [1]. We aim to mitigate this risk by relying on Web standards rather than bespoke APIs for expressing interactive software. 6) *Security and Privacy by default:* MyWebstrates delegates access controls to the underlying data substrate. For non-shared webstrates, this represents an increase in security and privacy, as data do not leave the browser. For shared webstrates, this currently represents a decrease in security as the underlying data substrate does not currently provide access controls. This is not a fundamental limitation; we expect it to be addressed in a future version of Automerge. 7) *You retain ultimate ownership and control* is partially fulfilled. While MyWebstrates live in the browser by default, and the user can selectively choose which webstrates to share with a sync server, Webstrates currently provides rudimentary per-webstrate tools. Assets and other dependencies add complexity to the meaning of just what is meant by *a* webstrate. Furthermore, once a webstrate has been shared, it is effectively beyond the creator's sole control to update or delete remote copies. However, this is a fundamental concern in decentralised content networks and not unique to MyWebstrates.

6.8 Other limitations

There are several limitations to the implementation of MyWebstrates. Offline collaboration over peer-to-peer is fragile to network interrupts and can be limited when used on networks enforcing strict client isolation. Furthermore, in the current implementation, as soon as two clients are peered, any open webstrates are shared, hence making it possible to inadvertently leak private content.

All document history is preserved by Automerge. Users cannot redact historical edits, beyond creating a copy with no history. Sensitive data such as passwords cannot easily be redacted and will thus be accessible to collaborators via the history APIs.

Storing assets in Automerge does not scale to very large assets such as, e.g., embedding a gigabyte video in a slideshow webstrate. Similarly, the service worker does not support streaming assets, which limits its use for, large assets such as video.

There is currently no user facing way to see what webstrates are stored in local storage in the browser, and if a user clears their browser cache, the data will be gone. Future work includes an interface to explore local webstrates and to export or persist them to disk, e.g., through the file system API [35].

Various aspects of the original Webstrates API have been omitted from MyWebstrates, e.g., user cookies[8] and messaging[9]. These features have had little use in software built on Webstrates, and will require being rethought for local-first software.

7 CONCLUSIONS

Webstrates presents a vision and a model of shareable, dynamic media in which users can freely extend and mold software to suit their own idiosyncratic needs, but it relies on a centralised server to support collaboration and sharing. We have introduced MyWebstrates, an extension to this vision that focuses on interoperability and on personal and collective sovereignty, in line with the original goals of Webstrates.

The MyWebstrates prototype provides a local-first solution that obviates the need for a webstrates server, pushing that role down to a lower-level data substrate with an optional sync server. This approach enables various configurations of offline use and greater control over whether webstrates are shared. By introducing a more general data substrate, we also facilitate interoperability between webstrates, other systems that share the same data substrate, and external systems, both on and off the web.

ACKNOWLEDGMENTS

This work has been partially funded by the Novo Nordisk Foundation (0086698), the Aarhus University Research Foundation (AUFF-E-2022-9-33), the Villum Foundation (VL-54492), and the French National Research Agency (ANR-21-ESRE-0030). Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies. We thank Janus Kristensen and Jonas Oxenbøll Petersen from CAVI for programming assistance and video editing. We thank Jason Kankiewicz for his work on Automerge-based USD support in Godot.

REFERENCES

[1] Steph Ango. 2023. File Over App. https://stephango.com/file-over-app. Accessed: 2024-03-28.
[2] Sriram Karthik Badam, Andreas Mathisen, Roman Rädle, Clemens N. Klokmose, and Niklas Elmqvist. 2019. Vistrates: A Component Model for Ubiquitous Analytics. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 586–596. https://doi.org/10.1109/TVCG.2018.2865144
[3] R. Ballagas, A. Szybalski, and A. Fox. 2004. Patch panel: enabling control-flow interoperability in ubicomp environments. In *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the.* 241–252. https://doi.org/10.1109/PERCOM.2004.1276862
[4] Santiago Bazerque. 2021. Hyper Hyper Space. https://www.hyperhyperspace.org/whitepaper. Accessed: 2024-03-28.
[5] Michel Beaudouin-Lafon. 2000. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems* (The Hague, The Netherlands). ACM, New York, NY, USA, 446–453. https://doi.org/10.1145/332040.332473
[6] Michel Beaudouin-Lafon. 2017. Towards Unified Principles of Interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 1, 2 pages. https://doi.org/10.1145/3125571.3125602
[7] Michel Beaudouin-Lafon and Wendy E. Mackay. 2000. Reification, polymorphism and reuse: three principles for designing visual interfaces. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces* (Palermo, Italy). ACM Press, 102–109. https://doi.org/10.1145/345513.345267
[8] Marcel Borowski, Janus Bager Kristensen, Rolf Bagge, and Clemens Nylandsted Klokmose. 2021. Codestrates v2: A Development Platform for Webstrates. (2021).

[8] https://webstrates.github.io/userguide/api/cookies.html
[9] https://webstrates.github.io/userguide/api/messaging.html

[9] Stephane Couture and Sophie Toupin. 2019. What does the notion of "sovereignty" mean when referring to the digital? *New Media & Society* 21, 10 (2019), 2305–2322. https://doi.org/10.1177/1461444819865984 arXiv:https://doi.org/10.1177/1461444819865984

[10] Liangrun Da and Martin Kleppmann. 2024. Extending JSON CRDTs with Move Operations. In *Proceedings of the 11th Workshop on Principles and Practice of Consistency for Distributed Data* (Athens, Greece) *(PaPoC '24)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3642976.3653030

[11] Peter Dalsgaard, Kim Halskov, and Clemens Nylandsted Klokmose. 2020. A study of a digital sticky note design environment. In *Sticky creativity*. Elsevier, 155–174.

[12] W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, and Trevor F. Smith. 2009. Experiences with recombinant computing: Exploring ad hoc interoperability in evolving digital networks. *ACM Trans. Comput.-Hum. Interact.* 16, 1, Article 3 (apr 2009), 44 pages. https://doi.org/10.1145/1502800.1502803

[13] Jonas Frich, Midas Nouwens, Kim Halskov, and Peter Dalsgaard. 2021. How Digital Tools Impact Convergent and Divergent Thinking in Design Ideation. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 431, 11 pages. https://doi.org/10.1145/3411764.3445062

[14] Joseph Gentle. 2011. ShareJS: Realtime collaboration editing in any app. https://github.com/josephg/ShareJS. Accessed: 2024-03-28.

[15] 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* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 3383–3392. https://doi.org/10.1145/1978942.1979446

[16] Danny Goodman. 1993. *The complete HyperCard 2.2 handbook*. Random House Inc.

[17] Jens Emil Grønbæk, Banu Saatçi, Carla F. Griggio, and Clemens Nylandsted Klokmose. 2021. MirrorBlender: Supporting Hybrid Meetings with a Malleable Video-Conferencing System. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 451, 13 pages. https://doi.org/10.1145/3411764.3445698

[18] 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* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 14, 14 pages. https://doi.org/10.1145/3586183.3606767

[19] Herb Caudill. 2024. Alice and Bob in wonderland: Bootstrapping identity and authority in a world without servers. https://herbcaudill.com/words/20240602-local-first-auth. Accessed: 2024-07-18.

[20] Ink & Switch. 2024. Patchwork lab notebook. https://www.inkandswitch.com/patchwork/notebook/. Accessed: 2024-07-23.

[21] Ink & Switch. 2024. Tiny Essay Editor. https://github.com/inkandswitch/tiny-essay-editor. Accessed: 2024-03-28.

[22] Kevin Jahns. 2024. Yjs: A CRDT framework for real-time collaboration. https://yjs.dev. Accessed: 2024-03-28.

[23] JsonML.org. 2024. JsonML. http://www.jsonml.org. Accessed: 2024-07-23.

[24] Alan Kay and Adele Goldberg. 1977. Personal dynamic media. *Computer* 10, 3 (1977), 31–41.

[25] Alan C Kay. 1996. The early history of Smalltalk. In *History of programming languages—II*. 511–598.

[26] Martin Kleppmann and Ink & Switch. 2024. Automerge: A JSON-like data structure that can be modified concurrently by different users, and merged again automatically. https://automerge.org. Accessed: 2024-03-28.

[27] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. 2019. Local-first software: you own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Athens, Greece) *(Onward! 2019)*. Association for Computing Machinery, New York, NY, USA, 154–178. https://doi.org/10.1145/3359591.3359737

[28] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proc. UIST '15* (Charlotte, NC, USA). ACM, 280–290. https://doi.org/10.1145/2807442.2807446

[29] 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* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 233–247. https://doi.org/10.1145/3332165.3347912

[30] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3173574.3173610

[31] Geoffrey Litt, Peter van Hardenberg, and Orion Henry. 2020. Project Cambria: Translate your data with lenses. https://www.inkandswitch.com/cambria. Accessed: 2024-03-28.

[32] Geoffrey Litt, Peter van Hardenberg, and Orion Henry. 2021. Cambria: Schema Evolution in Distributed Systems with Edit Lenses. In *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data* (Online, United Kingdom) *(PaPoC '21)*. Association for Computing Machinery, New York, NY, USA, Article 8, 9 pages. https://doi.org/10.1145/3447865.3457963

[33] Jiali Liu and James R Eagan. 2021. ADQDA: A Cross-device Affinity Diagramming Tool for Fluid and Holistic Qualitative Data Analysis. *PACM HCI: Proceedings of the ACM on Human-Computer Interaction* 5, ISS (2021), 19. https://doi.org/10.1145/3488534

[34] Matrix.org Foundation. 2024. Matrix.org: An open network for secure, decentralised communication. https://matrix.org. Accessed: 2024-03-28.

[35] Mozilla Developer Network (MDN). 2024. FileSystem API. https://developer.mozilla.org/en-US/docs/Web/API/File_System_API. Accessed: 2024-03-28.

[36] Mozilla Developer Network (MDN). 2024. IndexedDB API. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API. Accessed: 2024-03-28.

[37] Mozilla Developer Network (MDN). 2024. Service Worker API. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API. Accessed: 2024-03-28.

[38] Line Have Musaeus, Marie-Louise Stisen Kjerstein Sørensen, Blanka Sára Palfi, Ole Sejer Iversen, Clemens Nylandsted Klokmose, and Marianne Graves Petersen. 2022. CoTinker: Designing a Cross-device Collaboration Tool to Support Computational Thinking in Remote Group Work in High School Biology. In *Nordic Human-Computer Interaction Conference* (Aarhus, Denmark) *(NordiCHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 49, 12 pages. https://doi.org/10.1145/3546155.3546709

[39] Theodor Holm Nelson. 1995. The heart of connection: hypermedia unified by transclusion. *Commun. ACM* 38, 8 (1995), 31–33.

[40] Midas Nouwens, Marcel Borowski, Bjarke Fog, and Clemens Nylandsted Klokmose. 2020. Between Scripts and Applications: Computational Media for the Frontier of Nanoscience. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376287

[41] Pixar Animation Studios. 2021. OpenUSD. https://openusd.org/release/index.html. Accessed: 2024-03-28.

[42] DxOS Project. 2024. DxOS: Realtime local-first applications. https://dxos.org. Accessed: 2024-03-28.

[43] 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* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 715–725. https://doi.org/10.1145/3126594.3126642

[44] Replicache. 2024. Replicache: Realtime sync for any backend stack. https://replicache.dev. Accessed: 2024-03-28.

[45] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, Xavier Défago, Franck Petit, and Vincent Villain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 386–400. https://doi.org/10.1007/978-3-642-24550-3_29

[46] ShareDB. 2024. ShareDB: Realtime database backend based on Operational Transformation (OT). https://github.com/share/sharedb. Accessed: 2024-03-28.

[47] Guy Steele. 1990. *Common LISP: the language*. Elsevier.

[48] Chengzheng Sun and Clarence Ellis. 1998. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work* (Seattle, Washington, USA) *(CSCW '98)*. Association for Computing Machinery, New York, NY, USA, 59–68. https://doi.org/10.1145/289444.289469

[49] Daniel Supernault and Contributors. 2023. Pixelfed. https://pixelfed.org. Accessed: 2024-03-28.

[50] tldraw. 2024. tldraw. https://tldraw.com. Accessed: 2024-07-15.

[51] David Ungar and Randall B. Smith. 1987. Self: The power of simplicity. *SIGPLAN Not.* 22, 12 (dec 1987), 227–242. https://doi.org/10.1145/38807.38828

[52] Matthew Weidner, Martin Kleppmann, Daniel Hugenroth, and Alastair R. Beresford. 2021. Key Agreement for Decentralized Secure Group Messaging with Strong Security Guarantees. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2024–2045. https://doi.org/10.1145/3460120.3484542

[53] Ernest Wisniewski and Kbin contibutors. 2023. Kbin. https://kbin.pub. Accessed: 2024-03-28.

[54] World Wide Web Consortium (W3C). 2018. *ActivityPub*. W3C Recommendation. World Wide Web Consortium (W3C). https://www.w3.org/TR/activitypub/ Accessed: 2024-03-28.

[55] Brooklyn Zelenka, Daniel Holmgren, Irakli Gozalishvili, and Philipp Krüger. 2024. UCAN Specification. https://github.com/ucan-wg/spec. Accessed: 2024-03-28.