What Can Software Learn From Hypermedia?

Philip Tchernavskij[a], Clemens Nylandsted Klokmose[b], and Michel Beaudouin-Lafon[a]

a LRI, Univ. Paris-Sud, CNRS,
Inria, Université Paris-Saclay
F-91400 Orsay, France

b Digital Design & Information Studies,
Aarhus University
8200 Aarhus N, Denmark

Abstract Most of our interactions with the digital world are mediated by *apps*: desktop, web, or mobile applications. Apps impose artificial limitations on collaboration among users, distribution across devices, and the changing procedures that constantly occur in real work. These limitations are partially due to the engineering principles of encapsulation and program-data separation. By contrast, the field of hypermedia envisions collaboration, distribution and flexible practices as fundamental features of software. We discuss *shareable dynamic media*, an alternative model for software that unifies hypermedia and interactive systems, and *Webstrates*, an experimental implementation of that model. We give examples of patterns and challenges for software architecture that have emerged in our experimentation with *Webstrates*, and show that it subverts the principles of encapsulation and program-data separation.

Keywords Hypermedia, Human-computer interaction, Programming paradigms

The Art, Science, and Engineering of Programming

Perspective The Theoretical Science of Programming

Area of Submission Hypermedia, Human-computer interaction, Programming paradigms

1 Introduction

Modern personal computing is mediated by desktop, mobile, and web applications (*apps* for short). This model of software has become pervasive since the introduction of the Macintosh and Windows desktop environments.[1] The app model shows its limitation when confronted with the natural complexity of human activity: while it is common to add, remove or change collaborators, tools, procedures, or working materials in the physical world, these dimensions of flexibility are limited or nonexistent in computer-mediated activity. These limitations are especially evident as software has become ubiquitous in our social and professional lives.

In general, apps model procedures, in the sense that they couple what users can do, e.g., changing color, with what they can do it to, e.g., text. Some apps supports collaborating on a particular task, but limit the available tools and working materials. Some apps can be extended with new tools, but are highly specialized, e.g., for illustration or programming. Apps can only be combined in limited ways, e.g., it is possible to work on an image in two image treatment apps in turn, but impossible to compose those two apps to make the same tools available at the same time on the same image. We argue that these limitations are mainly due to the app model, because it models software as static systems isolated from each other.

By contrast, research on hypermedia has, since the early days, emphasized collaborative work, distributed access and changing activities. Hypermedia systems support knowledge work, i.e. activities typical in research, journalism, engineering, etc., which are now ubiquitous. Therefore we believe that hypermedia concepts can provide a sound basis to revisit and replace the app model.

After arguing our critique of apps, we review early work on hypermedia systems and present *shareable dynamic media*, a software model that unifies interactive systems and hypermedia to overcome the limitations of the app model.[2] Its main qualities are:

- Both *apps* and *documents* are expressed as what we call *information substrates*;
- All working material is part of a shared, interconnected medium;
- The structure of the medium can be accessed and changed by every user; and
- The medium is decomposable and recomposable.

2 Critiquing Software

The limitations of apps are inherent to the established model for building interactive software. Apps are made up of compiled programs, executed by operating systems. The structure of an app is embedded in a program that users have no ability to inspect or manipulate. As a result, software is made up of the *app*, which is static, and

[1] Note, however, that the Xerox Star, which spearheaded graphical user-interfaces, was document-based, not app-based.

[2] This paper draws heavily on the first author's master's thesis [**Tchernavskij2016**].

documents, which are dynamic.[3] This distinction is somewhat arbitrary: Most apps only let users change the files they work with, but some let users change the cosmetics of the app with stylesheets, or extend the app with new features through plugins.

Apps are tightly isolated from the environment in which they are used. To work on a document stored in a file, an app has to load the file and create an internal representation which it can change. This strictly limits how apps can be combined. Apps can be sequenced, i.e. a file output by one app can be loaded by another (if the formats are compatible), but they cannot concurrently work with the same file. Some apps share content through a remote database, but then bear the burden of maintaining consistency between the database and their internal state. This is more akin to a distributed app than an open environment.

The app model is the result of common architectures and software engineering principles that have good properties for engineers and developers, *but not for users*. From an engineering perspective, the app model is very reasonable: The separation between things that can change and things that cannot prevents users from messing them up, and allows expert designers to maintain a lot of control over how their software is used. Encapsulation means that each application can be developed with the assumption that it exists in a vacuum.

We argue that remodelling interactive software as a *medium* is a viable first step in finding practical alternatives to encapsulated, static apps and better addressing user needs. To uncover how such a software medium can subvert the limitations of apps, we turn to the early history of hypermedia.

3 The Early History of Hypermedia

In *Memory Machines*, Barnet recounts the early history of hypertext through its innovators, seminal ideas, and implementations. Barnet focuses on some landmark systems as "visions of potentiality" that were all eventually eclipsed by the World Wide Web [**Barnet2013**]. She defines hypertext as: "Written or pictorial material interconnected in an associative fashion, consisting of units of information retrieved by automated links, best read at a screen.".[4]

The goal of hypermedia is to allow people to work with information in all its natural complexity. In many real work situations, information is distributed, shared, and liable to change. A great example can be found in Tim Berners-Lee's proposal for what eventually became the World Wide Web:

> "Although [CERN is] nominally organised into a hierarchical management structure, this does not constrain the way people will communicate, and share information, equipment and software across groups. The actual observed working

[3] Web apps are partly inspectable, at least on the client side. However developers often obscure code through minifiers and obfuscators to circumvent the hypermedia aspects of the Web.

[4] Note that *hypertext* was never intended to only encompass textual content. We use the words *hypertext* and *hypermedia* interchangeably in this paper.

> structure of the organisation is a multiply connected "web" whose interconnections evolve with time." [**BernersLee1989**]

Hypermedia was introduced not as a subgenre of software, but as a different conceptualization of what software is. Ted Nelson coined the word *hypertext* [**Nelson1965**] and Doug Engelbart built the first functioning hypertext system, NLS (oNLine System) [**Engelbart1988**]. Both envisioned hypermedia as a way to apply computation to augment human memory and capability to manage complex information resources.

For example, NLS was used to manage project journals: interconnected public records of every program, document, note, and annotation made in the course of a project. The FRESS system [**VanDam1976**] developed at Brown University was used to create a shared workspace for literary analysis, connecting central texts with encyclopedic and historic resources, related texts, and threads of annotations. FRESS was applied in a poetry class where students read, analyzed, peer-reviewed, and discussed, all within the same interface. FRESS supported filtering documents to only show certain links based on a keyword search. In massively interconnected texts, this allows readers to only view the hypertext structure that is relevant to their current task.

While hypermedia systems are often reduced to *linking* documents together, an essential component of Nelson's hypermedia is *transclusion*, a mechanism to *compose* documents. A transclusion is a type of hypermedia link that *embeds* the same content at several locations, like a shared section in two encyclopedia articles. In Nelson's vision of Xanadu, every document is in practice composed of transclusions into a shared space of versioned documents. With transclusion, hypertexts move from a set of documents with embedded links referencing each other to a combinatorial space of possible documents built from the shared *docuverse* [**Nelson1987**].

To summarize, early hypermedia systems and research emphasized the following features, which are still relevant today:

- *Documents* are built out of a shared medium with rich interconnections to both manage and manipulate documents;
- Links can be used to join, navigate, group, compare, transclude and annotate documents.
- Systems should support the different roles and (information) needs of collaborators;
- Systems should not distinguish between authorship and consumption of media;
- Remixing existing material is a common task;
- To manage document volatility, systems should employ ubiquitous versioning.

Together, these features augment text (and other media) with low-level properties supporting collaboration, distribution and user appropriation. They form a sound basis to replace the concept of app with the concept of software as a medium.

4 From Hypermedia to Shareable Dynamic Media

Hypermedia approaches have been applied to software for general personal computing, e.g., Bill Atkinson's HyperCard [**Atkinson1987**] and the Smalltalk programming

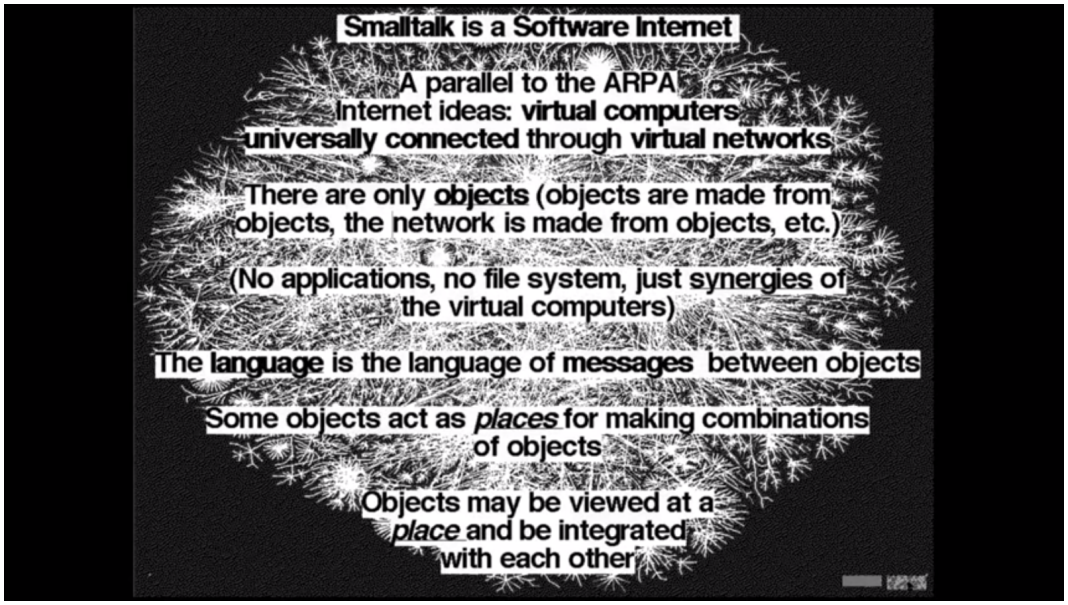

Figure 1 A frame from Kay's presentation of the Smalltalk system in his tribute to Ted Nelson [**Kay2014**].

language and system developed at Xerox PARC [**Kay1993**]. Each of these systems made a leap in abstraction by unifying the notions of *program* and *document*. Kay described the Smalltalk system as a "Software Internet" of objects extended and reconfigured by users (Figure **??**). HyperCard allowed users to construct simple utility software as interconnected *cards*: Documents consisting of pictures and text with embedded programs. Smalltalk and HyperCard users could remix and extend existing software, and share it with other users. Both the Smalltalk image and HyperCard stacks are versions of a distributed *docuverse* of software.

More recently, Klokmose et al. introduced *shareable dynamic media* [**Klokmose2015**], inspired by Kay and Goldberg's vision of *personal dynamic media* [**Kay1977**]. They introduce the concept of *information substrates* as "*... software artifacts that embody content, computation and interaction, effectively blurring the distinction between documents and applications. Substrates can evolve over time and shift roles, acting as what are traditionally considered documents in one context and applications in another, or a mix of the two. Substrates may be composed in various ways, e.g., one substrate can give meaning and structure to another.*". According to Klokmose et al., shareable dynamic media are collections of these substrates that are inherently *malleable* for appropriation by the user, *shareable* for collaboration between users, and *distributable* across computers of various sorts and sizes.

Interactive systems designed as shareable dynamic media are networks of information substrates mediating collaborative distributed interactions. These networks change over time as collaborators, tools and devices are added or removed. Transclusion is the main composition mechanism in these networks. For example, if two users want to work on a shared document, they both transclude the document in

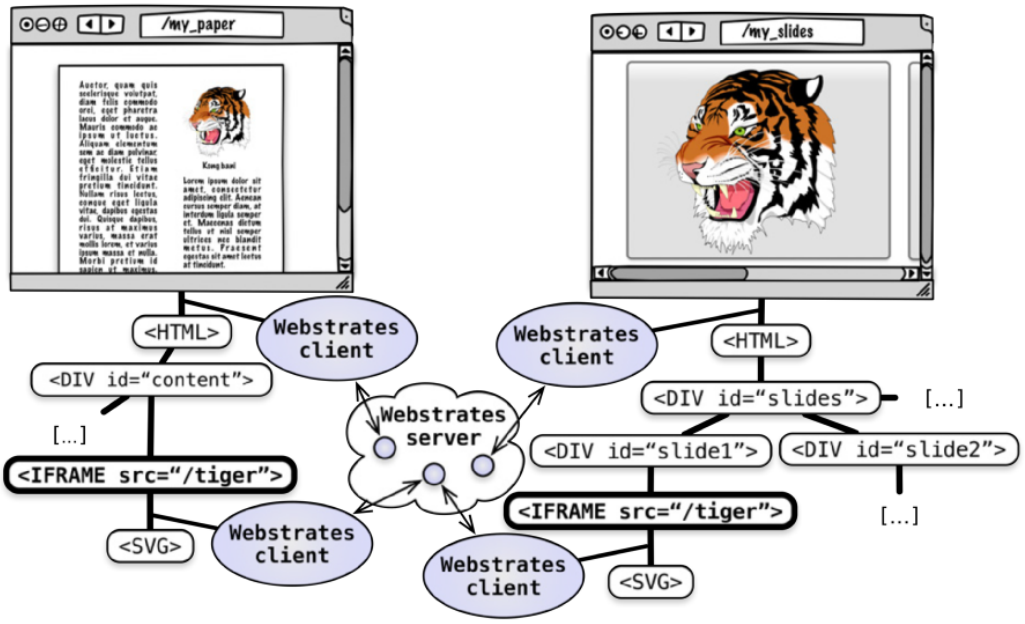

Figure 2 Two webstrates transclude the same image webstrate. Changes in the figure appear immediately in both windows. Reprinted from [**Klokmose2015**].

their respective reading/writing substrate. The reading/writing substrate is itself a document that can be shared and modified.

4.1 Webstrates

Webstrates [**Klokmose2015**] is a prototype implementation of shareable dynamic media built on top of modern web technology .[5] It consists of a webserver and a JavaScript client that turns any page served by the Webstrates server into a web substrate, or *webstrate*. Any client-side changes to the Document Object Model (DOM) of a webstrate, including embedded scripts and stylesheets, are persisted on the server and synchronized in real time with all other clients of the same webstrate using operational transformations [**ellis1991groupware**] on the DOM. Klokmose et al. [**Klokmose2015**] demonstrate how the *iframe* mechanism in HTML can be used to transclude one webstrate into another (Figure **??**), enabling the creation of application-document like relationships between two (or more) webstrates, and sophisticated mechanisms for software composition and reuse.

Three examples illustrate the power of this approach [**Klokmose2015**]:

- A writing webstrate is composed of tools for editing and formatting text, and transcludes any webstrate containing text as its document.

[5] There is a video demonstration and link to a public repository at www.webstrates.net

- A programming webstrate provides tools for editing scripts in other webstrates, and transcludes the writing webstrate as *its* document. It it the possible to share tools with other users, e.g., a citing tool to manage biblographic references.
- A slide-editing webstrate is used to create a presentation. It can use the same citation tool as the writing webstrate. The same presentation can then be transcluded by audience members for comments and questions and on the projector for presentation.

Webstrates also supports straightforward implementations of hypermedia mechanisms beyond transclusion. For example, bidirectional links and collaborative annotations are demonstrated in [**Bouvin2016**].

As illustrated by the Webstrates prototype, shareable dynamic media brings the previously enumerated qualities of hypermedia to general-purpose software. Shareable dynamic media:

- are distributed among people and devices. A system is not delimited by one person and one computer;
- can flexibly scale to add collaborators, tools and devices;
- blur the line between authorship and consumption of software. Each user can modify and extend their software without special-purpose tools.
- decouple documents from tools.

5 Making Shareable Dynamic Media Work

Based on our experience with shareable dynamic media in the Webstrates prototype, we now describe some architectural patterns that have emerged and research challenges for future work.

5.1 Experiences Designing Shareable Dynamic Software

Designing for shareability, malleability and distributability forces us to reevaluate the components we use to build software. We present three patterns that have emerged in our experiments with Webstrates: shared documents for asymmetric collaboration, reification to extend the notion of substrates, and shareable and distributable interaction instruments.

Shared documents Structured content formats such as HTML and SVG are good media for cooperation because they do not codify specific procedures to work with them and thus can easily be appropriated and reinterpreted. For example, collaborating users can transclude shared data in their preferred working contexts, as opposed to traditional apps, which are tightly linked to their document formats. This is illustrated in Webstrates with a collaborative system in which a shared document is edited by two users, applying different stylesheets [**Klokmose2015**]: one user can use a WYSIWYG-style editing webstrate, the other can directly edit the document as markup. We have also experimented with connecting different types of documents

together to support cross-domain collaboration. For example we have created a substrate that translates modifications of a vector graphics drawing into a mathematical graph and vice-versa [**Tchernavskij2016**]. Bidirectional transformations, e.g., bidirectional lenses [**Hofmann2012**], are a promising research area to address such needs for mapping document representations.

Extending the scope of the medium Part of the power of a new medium is its ability to represent content that it was not explicitly designed to support. We apply the interaction design principle of *reification* to extend the notion of first-class objects in a system [**Beaudouin-Lafon2000avi**]. For example, creating a local two-way binding between the state of a physical device and a document allows the device to be effectively treated as another type of document. For example, Klokmose et al. create a tangible interface for webstrates by linking a clock webstrate to a physical LEGO clock [**Klokmose2015**]. We have also reified input devices, to support cross-device interactions, and users, to support user awareness.

Interaction instruments Beaudouin-Lafon's instrumental interaction model [**Beaudouin-Lafon2000chi**] represents interactions as first-class objects, called *instruments*. Instruments encourage decomposing complex systems into component behaviors, and reusing interactions across domains. We have experimented with designing tools as instruments in Webstrates. An instrument typically encapsulates one or more commands in a component, which is decoupled from the document they are applied to. Our instrument pattern separates instruments into *functional elements*, which contain their logic, and *reification elements*, which contain their state and user interface. The reification elements transclude the function elements, so that each instance of an instrument refers to the same logic but has its own state. This pattern supports both sharing and distributing instruments: If a user wants to copy an instrument from one substrate to another, the user makes a copy of the reification element; If two users want to share an instrument, they transclude the same reification element. This lets users share instrument configurations, or perform cross-device interactions.

These emerging software components reflect Meyrowitz' 1989 critique of monolithic hypertext systems and Gat's critique of programming [**Meyrowitz1989**, **Gat2001**]. Their argument is that interesting and useful software should be built of small, dynamically interacting modules. Meyrowitz' critique is especially relevant because he argues that while hypermedia researchers have shown that their systems are useful, they have not seriously worked to enable those systems to propagate and co-exist in practice. We think that building shared dynamic media using the above patterns (among others), can address this problem.

5.2 Future Shareable Dynamic Media

Our experience with Webstrates has identified a set of limitations and challenges. Some, like granularity and centralization, are partly due to the choice of using the web as platform. Others, like programmable media and security models, are more

related to the underlying model of shareable dynamic media. All of them present, we believe, interesting research challenges.

Granularity The unit of links on the web is the URL, which refers to a location on a server containing a whole HTML document. In effect, the web is a "chunk" hypertext system. As a result, transclusion cannot be used to decompose a document into parts, but only to compose entire web pages. This makes appropriation of parts of a system difficult, and forces designers to establish the separable modules of a given system early. However, designing a proper way to identify and transclude parts of a document is an interesting question, especially when combined with versioning and real-time sharing.

Centralization Webstrates is deployed as a single web server synchronizing all its hosted documents. This allows it to subvert the same-origin policy implemented by web browsers, which prevents interaction between web pages originating on different servers. This centralized architecture limits interactions across documents on different servers, as well as offline use. A more distributed architecture is needed to support practical use on a large scale.

Programmable Media Webstrates models shareable dynamic media as markup documents with embedded scripts. But while Webstrates supports sharing document state across users and devices, the program state, held in the Javascript runtime, is not directly accessible and therefore cannot currently be shared. We need to explore better programming models for shareable dynamic media. For example, Basman et al. develop the notion of convergence between programs and their runtime representation as a measure of programming language liveness [**Basman2016**]. A live programming environment[6] may also help users understand and experiment with programming substrates. Simulations and games, among others, require real-time interactive physics engines. Modeling this type of component in a way that is shareable, malleable, and distributable remains an open research question.

Security models When users are free to inspect and change every aspect of the shared medium, it becomes impossible to guarantee security properties such as secrecy and authenticity. Webstrates implements basic read/write access controls and the ability to selectively turn off sharing of parts of a webstrate. Obviously, mechanisms must be devised to control the trade-off between openness, to let users appropriate the medium, and control, to avoid security breaches.

[6] e.g., Lively (http://lively-kernel.org/), Chorus (http://www.chorus-home.org/), or Boxer [**diSessa1986**].

6 Conclusion

The early hypermedia pioneers envisioned software as shared, distributed media, with which people could manage a world of complex, changing information. Shareable dynamic media is a software model inspired by systems such as HyperCard and Smalltalk that unifies the concepts of hypermedia and interactive software. The model acknowledges that computer-mediated activity does not happen in a vacuum, and that people break strict procedures and appropriate tools and content all the time. While the current model of apps breaks down in our world of ubiquitous and distributed computer-mediated activity, shareable dynamic media provides an alternative better suited to this expanded set of needs. By rethinking assumptions about the role of computing in human activity, we hope to inspire new research as well as critical responses.