Interactive Substrates for Malleable Software

James Eagan[a]

a LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Abstract In this position paper, I present work we have conducted around the goals of making software malleable. Malleable software is software that a computer operator can appropriate and adapt to suit their own situated, idiosyncratic needs in ways not explicitly envisioned by the software designer. I describe past work in this space conducted with a variety of collaborators and students, as well as historical influences in this space. I conclude with a hint at a direction we are currently exploring in which computation is embedded in composable substrates rather than articulated through code.

ACM CCS 2012

- **Human-centered computing** → **Interactive systems and tools**;

Keywords substrates, malleable software

The Art, Science, and Engineering of Programming

Perspective The Art of Programming

Area of Submission Social Coding, General-purpose programming

1 Introduction

Today's software is not soft. Worse, it is downright Orwellian in nature: as long as the operator follows the prescribed path intended by the software designer, it is easy to miss the cage that shapes and binds the way the user must consider and interact with the machine. Software designers create these tools with a specific purpose and context in mind. If the user should deign to stray from these uses, to use it in unintended ways, they will be hard-pressed to adapt it to their own idiosyncratic needs.

In early graphical user interfaces, customization was a common feature. Users would often adapt their environments to their own needs, alone or in collaboration, such as through customizing Unix dot-files [16] or their CAD environments [7]. Systems such as Buttons [18, 24] treat customizable widgets and their behaviors as first-class objects that users can shape to their own needs via their source code.

These are examples of malleable software. Malleable software is software that the user can appropriate, shape, and adapt to suit their own idiosyncratic needs [see, e. g., 14]. The malleability of these systems introduces its own set of tradeoffs: the system might be more difficult to learn, as each installation behaves slightly differently depending on how it is configured. These differences might thus hinder interoperability as well as the development of communities of practice [13]. Such software is intrinsically more fragile, as it is being used in ways not explicitly considered and tested by the developers. Moreover, such customizations may further be written by less-experienced developers or even by normal operators untrained in the ways of computer programming [see, e.g, 4]. Finally, the extent to which it is even possible to adapt and sculpt a given piece of software to one's own needs often depends on a combination of the intrinsic malleability of the software and the implicit and often hidden assumptions of the original designer.

2 Past Approaches

In Scotty [5], we took the approach of extending the Cocoa runtime with hooks to explicitly enable modifying existing applications without their source code, for example by modifying Apple's closed-source Quicktime Player to display external subtitles on DRM-encrypted videos. This approach relied on method swizzling and leveraging Objective-C's dynamic dispatch implementation (via the Python-Objective-C bridge). A Python-literate programmer could thus explore the running state of a program via various sensemaking tools (e.g., instruments to map rendered widgets to their underlying instances; class browsers to reveal their methods and inheritance structure; and an interactive Python prompt running in the program space). From that understanding, they could craft a plugin that would graft in a new behavior or override the existing behavior in an existing application.

With Shared Substance [8], we instead considered a programming model in which data and behavior are loosely-coupled, where different behaviors could be associated with the same underlying data. This approach enabled interesting multi-surface applications, where different devices could live-synchronize their underlying data,

but offer distinct behaviors suited to the interactive capabilities of the device or the user tasks they addressed.

In both of these approaches, malleability is possible, but presents a high barrier to entry. In Scotty, each customization is effectively an ad-hoc hack on an existing system. In Shared Substance, the system needs to be explicitly architected around its customization approach.

With Webstrates [11], we began to base such malleability around Web-based substrates, building on the core web technologies of the DOM, CSS, and Javascript. Each of these components provides an intertwined substrate that structures data (content), presentation, and behavior. By synchronizing the DOM and combining different aspects of each of these three distinct substrates, a Webstrate (itself a conceptual substrate built by composing these lower-level substrates) could provide a rich tool for exploring alternative ways of building complex, flexible interactive tools.

Mackay and Beaudouin-Lafon [17] introduce the notion of Interactive Substrates, which frame and structure these notions into which Webstrates fits. Interactive substrates provide a locus of interaction that serves to structure a user's objects of interest. They provide a set of constraints that guide that abstraction, that define what it means for an object to exist within that substrate. These substrates are further composable: higher level substrates can operate on a lower-level substrate to extend, refine, or constrain the substate's behavior.

Jiali Liu's dissertation work on ADQDA [15] demonstrates this flexibility for Affinity Diagramming for Qualitative Data Analysis. It provides a flexible (albeit not-very malleable) tool for qualitative analysis of interview transcripts. In it, each interview is stored in a rich-text substrate. Another substrate stores cross-linked snippets (or "codes" in thematic analysis terms). A "coding" substrate transcludes each interview and maintains a link to the snippets substrate, providing the user with an interface to dynamically create, remove, or re-code snippets. An affinity diagramming substrate further extends this underlying substrate to provide an alternative means for updating these codes via an affinity diagram. In this way, distinct substrates are combined in ways that provide for flexible analytic processes depending on the analyst's situation context (goals, mood, available technologies).

These tools, however, remain far from the goal of creating truly malleable software. In ADQDA, for example, the analyst is limited to combining the flexible but focused substrates provided by the system. Conceptually, new substrates could be proposed to support alternative analyses—but not by the user.

3 On Malleability

Our work with Webstrates aims to re-think the way that we build interactive software, starting with breaking down the fundamental notions around which it is built. The concepts of application and document, for example, prove to be limiting in ways that do not always cleanly map to modern software contexts, where "applications" and "processes" no longer cleanly align [6, 22]. An application might run on one or multiple computers, with one or multiple users, using zero, one or many input devices.

Webstrates is a vision and a prototype for a world where software is not built around notions of application and document but rather of substrates [17] and interactive instruments [1]. Documents are inherently dynamic and shareable, and the tools used to interact with them are meant to be malleable [11].

Other environments might not specifically create customization points, as above, but might be inherently malleable. Smalltalk [9], Lisp [20, 25], and Self [10], for example, make extensive use of dynamism where it is inherently possible to alter the behavior of a running program.

Each of these tools, however, is focused on programming as the means for molding the environment. Tools such as Boxer [3] push closer towards democratizing the ability to modify programs. It proposes a sort of a graphical programming language and structure, similar to the way that spreadsheets encourage end-user programming through its scaffolding methods [21].

4 The way forward

Our work with Codestrates [23], and further work by Borowski et al.$_{\text{on}}$ Varv [2], are steps toward this goal by creating programming environments explicitly built around notions of substrates. While they scaffold and support coding—in some ways similar to computational notebooks like Jupyter [12]—code remains the language of expression. (Web)substrates are merely concepts within this space that they leverage.

Substrates offer a promise, as yet unrealized, of being able to provide user-expressible artifacts for computation, that can help frame, structure, and scaffold the expression of that computation in complementary ways that go beyond code. For example, with June Bhartia, we are currently exploring ways to build interactive games using composable substrates. Rather than writing code (whether textually or visually [e.g., using blocks, 19]), domain experts express interactive behaviors through different kinds of substrates.

We are confident that interactive substrates can provide a rich means for all users to appropriate and adapt their software, whether they dream in code, merely view the machine as a tool to accomplish a goal, or fall somewhere in between.

References

[1] Michel Beaudouin-Lafon. "Instrumental interaction: an interaction model for designing post-WIMP user interfaces". In: *CHI 'oo: Proceedings of the SIGCHI conference on Human factors in computing systems*. The Hague, The Netherlands: ACM, 2000, pages 446–453. ISBN: 1-58113-216-6. DOI: http://doi.acm.org.www.library.gatech.edu:2048/10.1145/332040.332473.

[2] Marcel Borowski, Luke Murray, Rolf Bagge, Janus Bager Kristensen, Arvind Satyanarayan, and Clemens N. Klokmose. "Varv: Reprogrammable Interactive Software as a Declarative Data Structure". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI '22. 2022. DOI: 10.1145/3491102.3502064.

[3] Andrea A. diSessa and Harold Abelson. "Boxer: A Reconstructible Computational Medium". In: *Commun. ACM* 29.9 (Sept. 1986), pages 859–868. ISSN: 0001-0782. DOI: 10.1145/6592.6595. URL: http://doi.acm.org/10.1145/6592.6595.

[4] Brian Dorn and Mark Guzdial. "Graphic designers who program as informal computer science learners". In: *ICER '06: Proceedings of the 2006 international workshop on Computing education research*. Canterbury, United Kingdom: ACM, 2006, pages 127–134. ISBN: 1-59593-494-4. DOI: http://doi.acm.org.www.library.gatech.edu:2048/10.1145/1151588.1151608.

[5] James Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. "Cracking the cocoa nut: user interface programming at runtime". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. UIST '11. Santa Barbara, California, USA: ACM, 2011, pages 225–234. ISBN: 978-1-4503-0716-1. DOI: http://doi.acm.org/10.1145/2047196.2047226. URL: http://doi.acm.org/10.1145/2047196.2047226.

[6] James R. Eagan, Clemens N. Klokmose, and Eric Lecolinet. "What are Applications in Multi-surface Environments?" In: *Powerwall international workshop on ultra-high-resolution displays, CHI '13 Extended Abstracts*. ACM, Apr. 2013, page 6.

[7] Michelle Gantt and Bonnie A. Nardi. "Gardeners and gurus: patterns of cooperation among CAD users". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. Monterey, California, United States: ACM Press, 1992, pages 107–117. ISBN: 0-89791-513-5. DOI: http://doi.acm.org/10.1145/142750.142767.

[8] Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. "Shared substance: developing flexible multi-surface applications". In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pages 3383–3392. ISBN: 978-1-4503-0228-9. DOI: http://doi.acm.org/10.1145/1978942.1979446. URL: http://doi.acm.org/10.1145/1978942.1979446.

[9] Adele Goldberg and David Robson. *Smalltalk-80: the language and its implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983. ISBN: 0-201-11371-6.

[10] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[11] Clemens Klokmose, James Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. "Webstrates: Shareable Dynamic Media". In: *ACM Symposium on User Interface Software and Technology (UIST)*. Edited by ACM. UIST '15 Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. Charlotte, United States, Nov. 2015, pages 280–290. DOI: 10.1145/2807442.2807446. URL: https://hal.archives-ouvertes.fr/hal-01242672.

[12] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. "Jupyter Notebooks—a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), page 87.

[13] Jean Lave and Etienne Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, 1991.

[14] Geoffrey Litt, Josh Horowitz, Peter van Hardenberg, and Todd Matthews. *Malleable Software: Restoring user agency in a world of locked-down apps*. 2025. URL: https://www.inkandswitch.com/essay/malleable-software/ (visited on 2025-06-01).

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

[16] Wendy E. Mackay. "Patterns of sharing customizable software". In: *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*. Los Angeles, California, United States: ACM Press, 1990, pages 209–221. ISBN: 0-89791-402-3. DOI: http://doi.acm.org/10.1145/99332.99356.

[17] Wendy E. Mackay and Michel Beaudouin-Lafon. "Interaction Substrates: Combining Power and Simplicity in Interactive Systems". In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. CHI '25. New York, NY, USA: Association for Computing Machinery, 2025. ISBN: 9798400713941. DOI: 10.1145/3706598.3714006. URL: https://doi.org/10.1145/3706598.3714006.

[18] Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. "User-tailorable systems: pressing the issues with buttons". In: *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*. Seattle, Washington, United States: ACM Press, 1990, pages 175–182. ISBN: 0-201-50932-6. DOI: http://doi.acm.org/10.1145/97243.97271.

[19] David J. Malan and Henry H. Leitner. “Scratch for budding computer scientists”. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’07. Covington, Kentucky, USA: Association for Computing Machinery, 2007, pages 223–227. ISBN: 159593361I. DOI: 10.1145/1227310.1227388. URL: https://doi.org/10.1145/1227310.1227388.

[20] John McCarthy. “History of LISP”. In: *SIGPLAN Not.* 13.8 (Aug. 1978), pages 217–223. ISSN: 0362-1340. DOI: 10.1145/960118.808387. URL: https://doi.org/10.1145/960118.808387.

[21] Bonnie A. Nardi. *A small matter of programming: perspectives on end user computing*. MIT Press, 1993. ISBN: 0-262-1405305.

[22] Midas Nouwens and Clemens Nylandsted Klokmose. “The Application and Its Consequences for Non-Standard Knowledge Work”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI ’18. Montreal QC, Canada: Association for Computing Machinery, 2018, pages 1–12. ISBN: 9781450356206. DOI: 10.1145/3173574.3173973. URL: https://doi.org/10.1145/3173574.3173973.

[23] Roman Rädle, Midas Nouwens, Kristian Antonsen, James Eagan, and Clemens Klokmose. “Codestrates: Literate Computing with Webstrates”. In: *UIST ’17: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST ’17. Québec City, QC, Canada: ACM, 2017, pages 715–725. ISBN: 978-1-4503-4981-9. DOI: 10.1145/3126594.3126642. URL: http://doi.acm.org/10.1145/3126594.3126642.

[24] George G. Robertson, D. Austin Henderson Jr., and Stuart K. Card. “Buttons as first class objects on an X desktop”. In: *Proceedings of the 4th annual ACM symposium on User interface software and technology*. Hilton Head, South Carolina, United States: ACM Press, 1991, pages 35–44. ISBN: 0-8979I-451-1. DOI: http://doi.acm.org/10.1145/120782.120786.

[25] Guy L. Steele. “An overview of COMMON LISP”. In: *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*. LFP ’82. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1982, pages 98–107. ISBN: 0897910826. DOI: 10.1145/800068.802140. URL: https://doi.org/10.1145/800068.802140.

About the author

James Eagan is professor of Human-Computer Interaction. Contact him at james.eagan@telecom-paris.fr.
https://orcid.org/0000-0002-7107-7035