# What are Applications in Multi-surface Environments?

**James R. Eagan**
Télécom ParisTech &
CNRS LTCI UMR 5141
46, rue Barrault
75013 Paris, France
james.eagan@telecom-paristech.fr

**Clemens N. Klokmose**
Aarhus University
Dept. of Computer Science
Åbogade 34
DK-8200 Aarhus N, Denmark
clemens@klokmose.net

**Eric Lecolinet**
Télécom ParisTech &
CNRS LTCI UMR 5141
46, rue Barrault
75013 Paris, France
eric.lecolinet@telecom-paristech.fr

Copyright is held by the author/owner(s). *CHI 2013 Extended Abstracts*, April 27–May 2, 2013, Paris, France.
ACM 978-1-4503-1952-2/13/04.

## Abstract
Multi-surface environments, which may contain any combination of large, high-resolution display walls (powerwalls), interactive tabletops, fixed computing infrastructure, laptops, tablets, and smartphones all linked together, offer a rich opportunity for new interactions and collaboration. Creating applications for these environments, however, breaks many of the assumptions of traditional computing environments, and requires a redefinition of the meaning of what constitutes an application. Furthermore, they present challenges when faced with a heterogeneous environment consisting of both new multi-surface applications and legacy data and applications.

## Author Keywords
multi-surface interaction, applications, wall-sized displays

## ACM Classification Keywords
H.5.m [Information interfaces and presentation (*e. g.*, HCI)]: Misc.

## Introduction
Immersive rooms, such as the WILD Room [2], offer the exciting prospect of creating rich interactive information environments. Such a room may contain an ultra-high resolution wall-sized display (a powerwall), interactive

table-top displays, laptop computers, tablets, and smartphones. Additional interactions may be possible through active or passive 3D tracking systems, such as Vicon, which uses markers placed on tracking targets, or the Microsoft Kinect, which uses a recognition approach to detect targets. These may be connected through a high-bandwidth, low-latency wireless network.

A powerwall can have very different roles in an interactive environment. On one extreme is the powerwall as a synoptic display, *e. g.* displaying flight data in a flight control room or displaying stock trends in a stock exchange. Here, there might be little or no direct user interaction with the powerwall. At the other extreme, the powerwall is treated as a large electronic whiteboard for heterogeneous digital content (as exemplified in [5]). In the latter case, multiple people may interact with the wall at the same time, bringing content to and modifying content on the wall. This, in effect, results in a device that simultaneously hosts multiple types of content used by multiple people simultaneously—an approach that is very different from our traditional interactive devices that tend to be one user, one application at a time (figure 1).

**Figure 1:** Substance Canvas [5] running on a 16-computer wall-sized display and a tabletop surface. An application running on the laptop (bottom, right) is synchronized to the wall. A user (left) scales up a vector-based view of the application using a motion-tracked pointer.

Regardless of which devices are available in a particular interactive environment, our goal is to enable all of these devices to work seamlessly together, as if they were all designed for such an environment. We call these rooms *multi-surface environments*. One potential scenario for such a room, for example, involves a scientist who should be able to enter with the latest set of telescope imagery on her smartphone, slide it onto an interactive table using a pick-n-drop-like technique [6], and collaboratively triage those images around the table with her colleagues. When identifying an image for further analysis, she should be able to easily slide it onto the wall, showing a higher-resolution, higher-detail image. Furthermore, she should be able to run her traditional, laptop-based data analysis tools on the wall, alongside the raw imagery.

The above scenario raises several questions. What should the interaction look like? How does our brave scientist exchange her data between devices? Are they copied between devices? Cloned (*i. e.* shared between them)? What should happen if she receives a phone call and leaves the room? How should the system communicate these interactions so as to help her and her collaborators better predict them? Finally, how should applications present themselves in these kinds of environments? Does the scientist use the same applications she is used to from her laptop?

Each of our interactive devices has its strengths and weaknesses, and a powerwall is no different. If the interaction at the wall goes much beyond navigation to actual content production or editing, additional devices such as a personal computer or a tablet may be necessary. As a consequence, not only may the powerwall be used by multiple people, and contain heterogeneous content, but the "system" or "application" may span multiple devices in a dynamic fashion.

Not only is it challenging to design appropriate interactions and mental models for these kinds of envionments, but also it raises additional challenges for their implementation. Multi-surface environments are frequently dynamic: new devices may enter and leave the environment. They are also inherently distributed: interaction may span multiple devices. For example, interaction may use a smartphone and input controlled through a tracking system, computation may take place on a computing cluster in the room, and output may be split across the smartphone and wall. A specific

interaction might begin on one device and finish on another, as in pick-n-drop.

Multi-surface environments challenge the traditional notion of applications on several levels. Not only does it dilute the concept itself, but it also challenges *how* we actually build interactive software when distribution is the norm rather than the exception.

## Distributed Application Models

Various approaches have been taken to designing applications to run in these kinds of multi-surface envronments. Among them, the Stanford iRooms project [3], i-Land [7] and its corresponding BEACH model [8], and even our own Shared Substance architecture [5] aim to examine ways of enabling interaction across multiple devices in the same environment. These approaches offer different levels of abstraction on top of the details of network access, connectivity, and data synchronisation. They further propose different ways of coupling or de-coupling data and functionality, such that interacting with the same data on different devices may offer different styles of interaction depending on the particular device and on the current context.

All of these projects struggle with how to resolve the inherent departures of multi-surface environments from the traditional role of an application. In a standard, desktop-based setting, an application is relatively easy to define: a program process started by the user, such as by double-clicking an icon or by invoking a single command. That program may involve many processes, but they are all started by the same single action, and can logically be considered to be a part of the same program. We know how to think about these kinds of applications. If a user quits the application, its primary process and all of its associated processes shut down.

Mobile applications change this model some, but not by much. Quitting an application on a mobile device might not actually quit the application; it might only freeze its state but keep the process loaded and ready to go if the user returns back to it. In any case, the mental model of what happens is conceptually straight-forward.

In multi-surface applications, however, a logical application may involve processes running a variety of devices. For example, the Substance Canvas application (figure 1) involves a cluster of 16 machines powering a wall-sized display, a front-end scene graph server to coordinate them all, a tabletop display also synchronizing with the scene graph server, a Vicon tracking system relaying events to an input event server, and a wirelessly connected iPod Touch. This single logical application, thus, involves 21 distinct processes running on different devices. Most of these processes run as independent services, providing a part of an underlying infrastructure necessary to support the overall application. If the table were to shut down, for example, the other services would continue to operate without degradation. Should the input event server shut down, the scene graph would remain intact, with the system continuing to operate in a pure visualisation mode. The application itself is thus simultaneously the ensemble of all of these services running on all of these devices, but whose functionality depends on which subset of those devices happens to be running at the time.

In order to handle this ever-changing environment with its dynamic resources and constraints, the multi-surface applications we built with Shared Substance did not dictate interaction per se. Interaction was decoupled from

the data model and instead implemented as instruments [1] with the datastructure of the scene-graph as the interface. As such, these instruments could reside on any machine on the local network (figure 2).

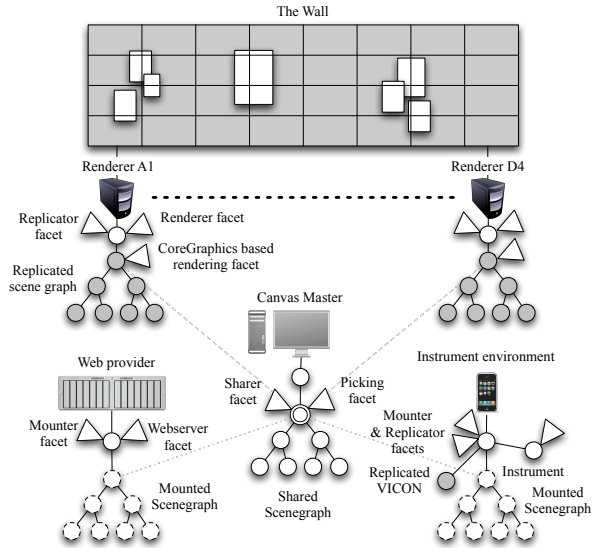


**Figure 2:** The architecture of the Substance Canvas (figure 1) running on a 16-computer wall-sized display, with a web-based data provider, a Vicon tracking provider, and an iPod Touch input device.

The Shared Substance project explores an underlying middleware model for creating multi-surface applications, but it only provides a starting step. The instrumental interaction model that we used for interaction, coupled with Shared Substance's data-oriented programming model [5] proves useful for these kinds of environments, but still requires nuanced interactions with the underlying framework. Ultimately, it needs to be easier for programmers to create their own interactive instruments and for users to find, collect, and use them.

## User Data in Multi-surface Environments

One of the real strengths of multi-surface environments is their rich, heterogeneous display, computational, and interactive properties. In order to take advantage of the richness of these environments, we want it to be easy for a user to bring his own data into the environment and to manipulate it. Interaction techniques such as pick-n-drop [6] are one elegant solution, but are surprisingly tricky to implement. We added different data providers to the shared environment, each of which could export data into the shared scene graph. We added data providers for email, web, anoto pen, and even a printer interface. As such, a user could email an image, upload an SVG document, scribble annotations on a piece of paper, or print a PDF document to the wall. With interactive instruments, he could then move, organise, scale, and otherwise manipulate these objects.

One seductive use of such wall-sized displays, however, is collaboration with applications running on a user's device. Beyond importing individual documents into the environment, one may wish to use a specialized application in the environment. For example, the astronomer in the earlier scenario may wish to use an analysis tool on her laptop to anotate different parts of the sky revealed in the latest imagery. But simply mirroring the window on her laptop on the wall is insufficient. On a high resolution wall, the window may potentially appear at a *smaller* physical size than is displayed on the lower-resolution laptop screen. Stretching that window on the wall results in "chunky pixels" due to the rasterized source image. Stretching the window beyond the size of the laptop's integrated display is typically not possible

with most windowing environments and would further require extensive memory and graphics resources to drive an ultra-high resolution wall-sized display.

We've experimented with this using the Scotty [4] toolkit, which provides deep hooks into Apple's Cocoa framework, thus allowing access to a vector graphics representation of an application window. We used this in conjunction with Substance Canvas to "beam" a live window from a laptop to the wall (figure 1). Because Scotty has access to the underlying window drawing operations, it can reconstruct a vector-based PDF of the window on the wall. Thus, vector-based content, such as text, shapes, and paths, scales up smoothly. Raster-based content, such as images, still scales up as chunky pixels, but at the backing resolution, which may be higher than the screen resolution. As the application updates its display on the laptop, so too does its image on the wall. We can further map pointing events back to the source application, allowing a two-way interaction between the laptop and wall. A yet unsolved challenge is how to enable interaction with the window on the wall using the interaction techniques for the wall in a sensible manner.

Ultimately, Scotty is a hack. It works because we were able to integrate deep inside the Cocoa framework to tap low-level drawing operations. While this approach is able to sustain drawing at interactive or near-interactive speeds, it is slow and fragile. A longer-term goal is examine how we can re-design desktop-based applications to better co-exist and appropriate such environments. We are beginning to see some tools for this kind of capability with audio and video files in Apple's AirPlay system to a dedicated hardware device (Apple TV) or from Samsung's smartphones and tablets to their latest TVs. These tools provide a rudimentary capability that does not allow for two-way interaction, let alone adapting the interaction to the particular configuration.

## Some (grand) challenges

*The role of applications*

We posit that powerwalls rarely make sense as isolated devices. They will most likely be part of a multi-surface environment where devices may dynamically come and go. The grand challenge is thus to rethink the notion of applications as something outwards rather than inwards pointing. Historically, it has been straight-forward to delimit the boundaries of an application. While office suites, for example, start to press upon these limits, it is still relatively easy to argue where an application begins and ends. A multi-surface application involves a collection of independent but autonomous processes all working together to provide a what a user might call an application. But how do we conceptualize the extent and boundaries for the user? And for the developer?

*Integrating with traditional software*

Powerwalls and multi-surface environments should not force us to reinvent the wheel. The software for our personal devices has matured and gained a foothold over the decades. However a powerwall is not a personal device. It is a part of a typically shared environment, hence a challenge is how to adapt this wall as a collaborative artifact that operates in conjunction with existing data and applications. Furthermore, how can we incorporate traditional applications that were designed with one user in mind. A naïve approach might be to abandon traditional applications as legacy, but even so, moving forward there will continue to be a divide between the use of applications in a multi-surface, powerwall-enabled setting and in a single-user, single-device context. For example, how can we bridge

between novel interaction techniques with touch or motion capture to applications designed for a mouse and keyboard?

## Acknowledgements

This work is funded in part by the Région Île-de-France/Digitéo and the Carlsberg Foundation. We thank our collaborators on the WILD and DigiScope projects, especially Michel Beaudouin-Lafon and Wendy Mackay.

## References

[1] M. Beaudouin-Lafon. 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*, 446–453, New York, NY, USA, 2000. ACM.

[2] M. Beaudouin-Lafon, S. Huot, M. Nancel, W. Mackay, E. Pietriga, R. Primet, J. Wagner, O. Chapuis, C. Pillias, J. R. Eagan, T. Gjerlufsen, and C. Klokmose. Multisurface interaction in the wild room. *Computer*, 45(4):48 –56, april 2012.

[3] J. Borchers, M. Ringel, J. Tyler, and A. Fox. Stanford interactive workspaces: a framework for physical and graphical user interface prototyping. *Wireless Communications, IEEE*, 9(6):64–69, dec. 2002.

[4] J. R. Eagan, M. Beaudouin-Lafon, and W. 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, 225–234, New York, NY, USA, 2011. ACM.

[5] T. Gjerlufsen, C. N. Klokmose, J. Eagan, C. Pillias, and M. Beaudouin-Lafon. Shared substance: developing flexible multi-surface applications. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, 3383–3392, New York, NY, USA, 2011. ACM.

[6] J. Rekimoto. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, 31–39, New York, NY, USA, 1997. ACM.

[7] N. A. Streitz, J. Geißler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz, and R. Steinmetz. i-land: an interactive landscape for creativity and innovation. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '99, 120–127, New York, NY, USA, 1999. ACM.

[8] P. Tandler. The beach application model and software framework for synchronous collaboration in ubiquitous computing environments. *J. Syst. Softw.*, 69(3):267–296, Jan. 2004.