Instrumental Interaction in Multisurface Environments

Michel Beaudouin-Lafon

Instrumental Interaction in Multisurface Environments

Position paper for the TEI 2015 workshop on Interactive Infrastructures

Michel Beaudouin-Lafon
Université Paris-Sud, France
mbl@lri.fr

For the past few years, we have been working on developing infrastructure and applications for interactive rooms that include large, wall-sized displays, tabletops, laptops, tablets and other interaction devices, such motion tracking devices. Our goal is to create the next generation of interactive systems that will support collaborative, distributed interaction over a range of devices and environments. We have worked extensively with "extreme" users who need such environments to deal with large and complex datasets, using participatory design to understand their need and prototype solutions.

Our WILD[1] room [5] features a 130-million pixel wall-sized display made of 32 30" monitors and powered by a 16-computer cluster, a 10-camera VICON motion tracking system, an FTIR tabletop and a variety of devices such as tablets, smartphones and wireless pointing devices. WILD is part of a network of 10 interactive rooms developed by the Digiscope project. All rooms feature large display surfaces and rich interaction devices. Two are immersive CAVEs; the others feature power walls with varying resolutions and sizes. Some have 3D capability; others are touch-enabled. The 10 rooms are interconnected with a telepresence system designed to support rich remote collaboration.

This unique infrastructure provides us with a rich testbed for experimenting with interaction techniques and interaction models for distributed interfaces. It also raises challenging technical issues for environments where devices interoperate smoothly and where configurations can be easily defined, stored and recalled.

Supporting Distributed Interaction

In order to support a variety of input devices in such environments, we have experimented with a variety of solutions. We began with the OSC protocol as the *lingua franca* for devices and applications. Although very flexible, thanks in particular to the many implementations of OSC, we found it difficult to aggregate data from multiple devices into coherent interactions. For example, we use the VICON tracking system to locate a device, such as a tablet, in the room, in order to use it as a laser pointer to designate objects on the wall display. The tablet features a touch-based user interface to manipulate a designated object. Where should we coordinate the information from the tracking system and the tablet to provide input to the application? In the tablet? In the application? In a third-party component?

We found that using a dedicated piece of software to aggregate input from a variety of devices and send commands to the application(s) was the best solution. We created the WILD Input Server [11] (figure 1), based on the ICon Input Configurator [5]. The WILD Input Server can receive input through a variety of input protocols such as OSC, TUIO, VRPN, BlueTooth. Users can interactively create visual, data-flow oriented model allows that process device input and output commands to application through OSC or Web sockets. This solution decouples the logic of input management from the applications, making it easy to prototype and test different solutions without touching the application.

---

[1] WILD stands for Wall-Sized Interaction with Large Data

The ICon data-flow model is well-adapted to managing input devices at a low level. We found it to be especially powerful when combined with a state-based model that describes interaction at a higher level. We developed the SwingStates [1] toolkit, which extends the Java language with state machines, and combined it with the data-flow model in ICon to create FlowStates [2], which is used in the WILD Input Server to create new modules. The result is a powerful infrastructure for managing distributed input.

Developing applications for such environments is a challenge because the execution environment is so different from regular desktop applications that we typically need to develop applications from scratch. At the same time, users expect to be able to use the applications they know, and we want to save on development costs. We have experimented with *code injection* to leverage existing applications in distributed environments. In Scotty [7], we inject Python code into Mac OS Cocoa applications to hijack both input and output. This lets us control a Mac OS application from remote devices, and manipulate its output to, e.g., display it remotely. In Hydrascope [9], we inject code into existing Web applications using Chrome extensions, to achieve similar effects. While these solutions are rather ad hoc, they provide useful insights into the requirements for making applications more independent from their execution environment, and we hope to be able to provide guidelines and frameworks for the development of future applications.

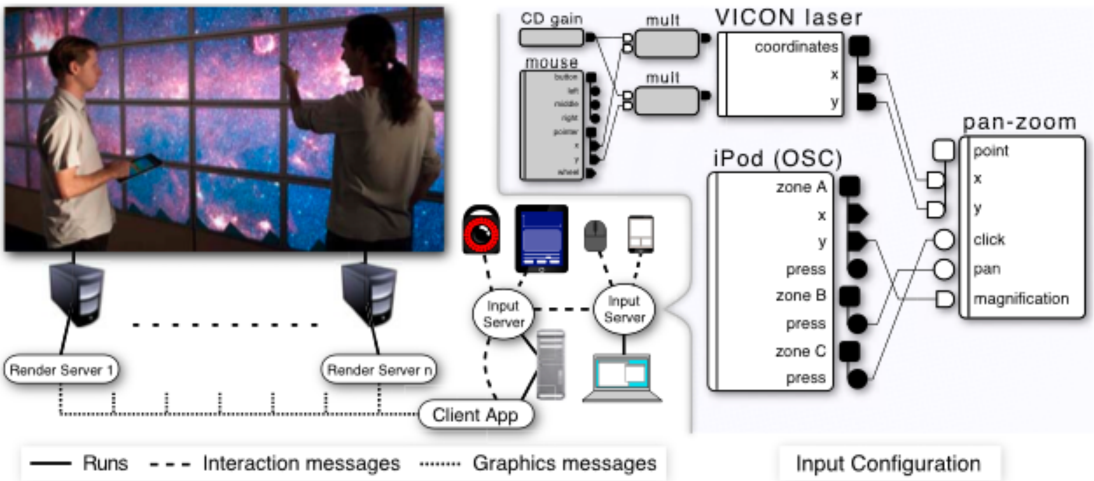

Figure 1 – The WILD Input Server: the configuration on the right describes the pan-and-zoom tool used on the left on the WILD wall-sized display.

Distributed Instrumental Interaction

Sorting out the technical issues of gathering input, redirecting output and wiring together heterogeneous devices with existing applications addresses only part of the problem of creating distributed interactive systems. We also need to understand how to ensure that interaction is consistent, discoverable and appropriable by users. Our approach is to develop a sound *interaction model* that guides designers by providing a conceptual model that is both easy to understand and powerful.

We developed Instrumental Interaction [3] based on the observation that our interaction with the physical world is most often mediated by tools and instruments: We use a pen to write, a switch to turn on a light, a car to travel. In fact, tool-building is a distinguishing characteristic of the human race which, with language, gave us an evolutionary advantage. We also often appropriate physical tools to use them in ways they were not designed for, such as using a knife to tighten a screw. While interactive software usually lacks this kind of flexibility, instrumental interaction lets us thinks of instruments and their affordances independently of the objects they manipulate.

Thinking in terms of interaction instruments has proven a powerful way to design interfaces, and we have identified design principles [4] to help generate instruments: reification helps identify new instruments, polymorphism helps create more powerful instruments, and reuse helps leverage existing objects and instruments. Together, these three principles have proven powerful as a generative tool for instrumental interaction.

More recently, we have adapted instrumental interaction to multi-surface environments [10] and created a software environment based on these principles [8] (figure 2). We have used this approach to create applications used by scientists and we continue to develop both the principles and the technical infrastructure to create more powerful distributed interactive systems. Figure 3 shows the canvas application of figure 2 with the underlying distributed architecture.

I am interested in participating in this workshop both to bring to the table our experience in developing distributed interfaces and to learn from others. I believe the time is ripe to come up with shared principles and infrastructure to support such interfaces.

Figure 2 – Two applications developed with Shared Substance [8]: a visualization application for collections of 3D brain scans, and a canvas to present and manipulate documents across multiple surfaces.

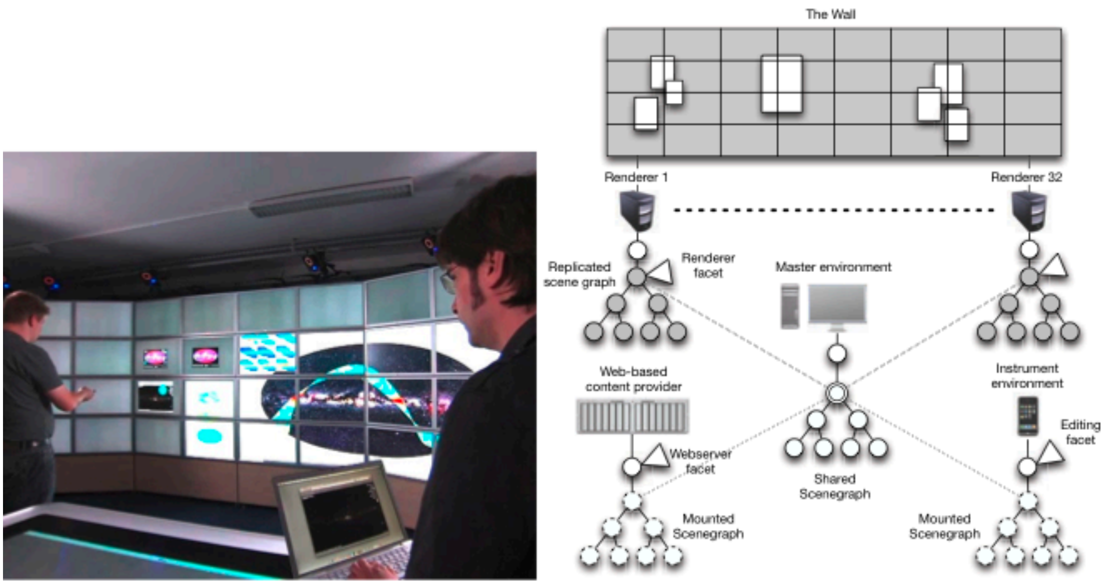

Figure 3 – The distributed architecture underlying the canvas application, based on a sharing objects across the components running on the 20+ computers involved in the room.

References

1. Caroline Appert and Michel Beaudouin-Lafon. (2008). SwingStates: Adding state machines to Java and the Swing toolkit. *Software Practice and Experience*, 38(11), 1149-1182, Sep 2008.
2. Caroline Appert, Stéphane Huot, Pierre Dragicevic and Michel Beaudouin-Lafon. FlowStates : Prototypage d'application interactives avec des flots de données et des machines à états. In *Proc. 21ème conférence francophone sur l'Interaction Homme-Machine*, IHM 2009, 119-128. ACM, 2009.
3. Michel Beaudouin-Lafon. Instrumental Interaction: an inter- action model for designing post-WIMP user interfaces. In *Proc. ACM Conference on Human Factors in Computing Systems (CHI'2000)*, 446–453. ACM, 2000.
4. Michel Beaudouin-Lafon and Wendy E. Mackay. Reification, polymorphism and reuse: three principles for designing visual interfaces. In *Proc. ACM Conference on Advanced Visual Interfaces (AVI'2000)*, 102–109. ACM, 2000.
5. Michel Beaudouin-Lafon, Olivier Chapuis, James R. Eagan, Tony Gjerlufsen, Stéphane Huot, Clemens Klokmose, Wendy Mackay, Mathieu Nancel, Emmanuel Pietriga, Clément Pillias, Romain Primet, Julie Wagner. Multi-Surface Interaction in the WILD Room. *IEEE Computer*, vol 45, nº 4, pp. 48-56, 2012.
6. Pierre Dragicevic and Jean-Daniel Fekete. Support for input adaptability in the Icon toolkit. In *Proc. Multimodal Interfaces*, ICMI '04, 212–219. ACM, 2004.
7. James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. Cracking the cocoa nut: user interface programming at runtime. In *Proc. User Interface Software and Technology*, UIST '11, 225–234. ACM, 2011.
8. Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Cle ́ment Pillias, and Michel Beaudouin-Lafon. Shared Substance: developing flexible multi-surface applications. In *Proc. Human Factors in Computing Systems*, CHI '11, 3383–3392. ACM, 2011.
9. Björn Hartmann, Michel Beaudouin-Lafon, Wendy E. Mackay. Hydrascope: Creating Multi- Surface Meta-Applications Through View Synchronization and Input Multiplexing. In *Proc. 2nd ACM International Symposium on Pervasive Displays*, Mountain View, United States. 43-48, 2013.
10. Clemens Klokmose and Michel Beaudouin-Lafon. VIGO: Instrumental interaction in multi-surface environments. In *Proc. Human Factors in Computing Systems*, CHI '09, 869–878. ACM, 2009.
11. Emmanuel Pietriga, Ste ́phane Huot, Mathieu Nancel, and Romain Primet. Rapid development of user interfaces on cluster-driven wall displays with jBricks. In *Proc. Engineering Interactive Computing Systems*, EICS '11, 185–190. ACM, 2011.