Collaboration Strategies for Drag-and-Drop Interaction with Multiple Devices

Stephen Hughes(✉), Marc Davenport, and Dalton Ott

Coe College, Cedar Rapids, USA
{shughes,mdavenport,ddott}@coe.edu

Abstract. ManyMouse is a software tool that revisits the ability of multiple users to collaborate by connecting their personal independent mouse hardware to a shared computer. This tool, implemented at the system level, assigns each input device to its own simulated mouse cursor, extending the potential for collaboration to any application. ManyMouse is currently being used as a platform to explore various coordination strategies and assess their impact on learning potential. Previous work on this topic has largely focused on inferring a group's intention from the relative positioning of the mouse pointers. This work attempts to extend those ideas to include coordination of direct actions such as drag-and-drop.

Keywords: Single display groupware · Collaborative input · Collective interaction · Mice · Drag-and-drop

1 Introduction

Single Display Groupware systems have demonstrated benefits in educational settings by enriching collaboration opportunities, opening communication channels among participants, and encouraging peer-teaching [1]. Numerous studies have explored a range of different approaches for managing collaborative input – from enforced turn taking while sharing a single mouse [2] to simultaneous control of a single mouse pointer [3]. The vision for single display groupware calls for each user to have their own private, independent input channel, but this is at odds with the way that operating systems behave when multiple devices are attached. Typically, inputs generated by movements from multiple mice are aggregated to adjust the position of a single on-screen mouse pointer. Therefore, when users wish to operate their own physical hardware in a collaborative setting, specialized software, such as described in [4, 5], is needed to create and manage multiple on-screen pointers. With this software in place, it is relatively easy to recognize and respond to basic group behaviors such as "swarming" or "convergence" which are based strictly on the position of multiple mouse pointers. However, more complex behaviors, such as synchronizing clicks and drag-and-drop operations have proven more difficult [5]. In order to capitalize on the benefits of multi-device collaboration with popular drag-and-drop programming environments such as Scratch, MIT AppInventor and other Blockly-based languages, it is important to develop strategies and recognize behaviors that can support this level of interaction.

2 ManyMouse

2.1 Design

Like other software solutions, ManyMouse is a tool that enables multiple users to have control of their own independent mouse cursor in a shared environment. However, this tool was explicitly designed to serve as a test bed for various interaction strategies to support drag and drop operations. As shown in Fig. 1, ManyMouse intercepts raw input coming from any number of connected mice and filters the input according to the state of the mouse. This allows the program to not only determine how to display and position a customized mouse icon, but also affect the behavior of the device by deciding which system messages to pass through to the operating system.

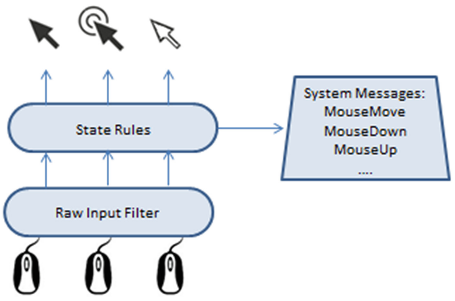

Fig. 1. ManyMouse schematic

Problems arise from the fact that “dragging” is not an explicit action in most systems; rather it is a derived from a temporal series of system messages. The system understands that it is dragging if it detects a MouseDown message followed by a sequence of MouseMove events; it exits the dragging state when it receives a MouseUp message. Without the intervention of a tool like ManyMouse, most systems will not discriminate the origin of system messages. Therefore one user’s attempt to drag an icon can be routinely interrupted by another user clicking; the second user’s MouseUp message forces the first user to drop their icon. However, solving this problem is not as simple as associating system messages with a specific input device; drag-and-drop problems arise from both the system level and the user interaction level.

2.2 Strategies

The ManyMouse system provides the flexibility to implement a variety of interaction strategies by simply altering the state rules. At the most basic level, users might be allowed to designate a “primary” device which is capable of performing unrestricted actions, while other devices are allowed to perform only actions that don’t interfere with the primary device’s actions. This delegates authority to the user of a particular device to act on behalf of the group. This technically works as a system-level solution, however, it can be problematic if authority is given to a domineering personality who doesn't ever want to relinquish control. Participants with limited capabilities are likely to feel alienated and disengaged and will not find value in this kind of system. Another strategy might attempt to democratize the role of the primary device by allowing access to sensitive actions to be granted on a First-Come, First-Serve basis. With respect to drag-and drop operations, this strategy can be implemented by simply ignoring all MouseDown and MouseUp events from other devices until an initial MouseDown event is completed with a corresponding MouseUp event from the same device. Numerous other strategies (such as turn-taking or timeouts) for ManyMouse can be imagined and implemented as a set of state transitions. From a user experience perspective, strategies that are both transparent and equitable are likely to be more successful.

3 Collective Interaction

One intriguing variation for managing collaborative drag-and-drop activities is known as Collective Interaction – a term coined by [6] to describe the natural phenomenon of individuals deliberately coordinating their input to achieve more complicated tasks. For example, when moving a heavy piece of furniture; there is an active dialogue about where to lift, how quickly to move or if a break is needed. In this paradigm, the operators must focus their attention on the input that is needed to produce a result and its rationale rather than strictly on the end result. In the context of drag-and-drop interfaces, elements may be perceived as 'too heavy' for one user to manipulate by themselves; they would need to coordinate with a peer to collectively manipulate the object. This constant negotiation of user input may prove quite valuable in a collaborative setting. As the collaborators attempt to collectively come to agreement on a course of action, individuals would need to justify their strategies in attempts to persuade others to participate in the collective behavior.

To implement collective drag-and-drop in ManyMouse requires the implementation of six states, as outlined in Fig. 2, with transitions outlined in Table 1.

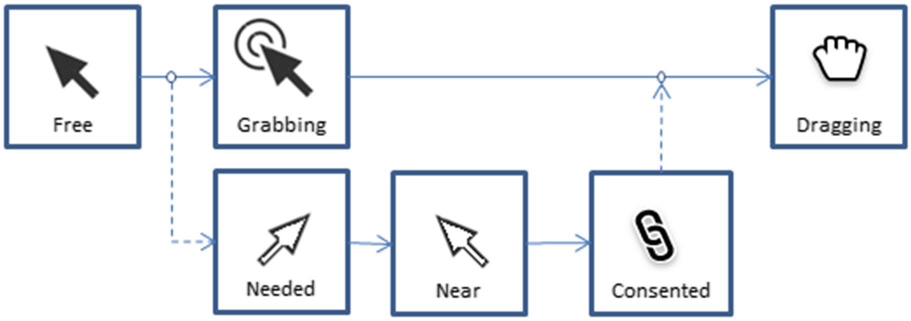

Fig. 2. Collective drag-and-drop states

Table 1. State transitions

State	MouseDown	MouseUp	MouseMove
Free	Grabbing; others to needed	Ignored	Passed through
Grabbing	N/A	Free (all)	Ignored
Needed	Ignored	Ignored	Passed through
Near	Consented	Ignored	Passed through
Consented	N/A	Near	Ignored
Dragging	N/A	Free (Drop)	Passed through

The intuition for this strategy is as follows. All pointers are considered "Free" until the first mouse button is pressed. If the button is immediately released, the act is counted as a "click" and all pointers are Free again. However, if the user holds the button down, it indicates that they have grabbed an icon, and need to wait in that location until some collaborators come to help to drag it elsewhere. Whenever another user comes to help they can agree to the actions of the "dragger" by holding down their mouse button. However they can withdraw their consent at any time by simply releasing the mouse button. The position of the consenter's mouse pointer moves relative to the dragging pointer; this eliminates the need to synchronize movements. The implication is that the dragger ultimately decides where to drop, but they must have the consent of their collaborators to perform any repositioning.

4 Evaluation

A functional prototype of ManyMouse has been developed and tested with the first-come, first-served and collective interaction strategies. Initial and informal evaluations of the system continue to suggest refinements before a formal evaluation can be performed. However, the design for that evaluation is currently being explored and expected to be conducted in the near future.

References

1. Stewart, J., Bederson, B., Druin, A.: Single display groupware: a model for co-present collaboration. In: CHI 1999 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, pp. 286–293 (1999)
2. Inkpen, K., Booth, K.S., Klawe, M., McGrenere, J.: The effect of turn-taking protocols on children's learning in mouse-driven collaborative environments. In: Graphics Interface, pp. 138–145 (1997)
3. Hughes, S., Bardell, C., Schafer, J.B.: Human performance with multiple devices influencing a single cursor. In: Human Factors and Ergonomics Society 57th Annual Meeting, San Diego, CA, pp. 808–812 (2013)

4. Moraveji, N., Inkpen, K., Curtrell, E., Balakrishnan, R.: A mischief of mice: examining children's performance in single display groupware systems with 1 to 32 mice. In: CHI 2009 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, pp. 2157–2166 (2009)
5. Heimerl, K., Vasudev, J., Buchanan, K., Parikh, T., Brewer, E.: MetaMouse: improving multi-user sharing of existing educational applications. In: ICTD 2010 Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development, New York (2010)
6. Krogh, P., Petersen, M.: Collective interaction: let's join forces. In: COOP 2008 (2008)