FlowBlocks: A Multi-Touch UI for Crowd Interaction

Florian Block[1]*, Daniel Wigdor*[2]*, Brenda Caldwell Phillips*[1]*, Michael S. Horn*[3]*, Chia Shen*[1]

[1]Harvard University, [2]University of Toronto, [3]Northwestern University
[1]{fblock | cshen | phillips}@seas.harvard.edu, [2]dwigdor@dgp.toronto.edu,
[3]michael-horn@northwestern.edu

ABSTRACT
Multi-touch technology lends itself to collaborative crowd interaction (CI). However, common tap-operated widgets are impractical for CI, since they are susceptible to accidental touches and interference from other users. We present a novel multi-touch interface called *FlowBlocks* in which every UI action is invoked through a small sequence of user actions: dragging parametric *UI-Blocks*, and dropping them over operational *UI-Docks*. The FlowBlocks approach is advantageous for CI because it a) makes accidental touches inconsequential; and b) introduces design parameters for mutual awareness, concurrent input, and conflict management. FlowBlocks was successfully used on the floor of a busy natural history museum. We present the complete design space and describe a year-long iterative design and evaluation process which employed the Rapid Iterative Test and Evaluation (RITE) method in a museum setting.

Author Keywords
Crowd Interaction, Multi-Touch UI, Drag & Drop.

ACM Classification Keywords
H5.m. Information interfaces and presentation: User Interfaces. – Graphical User Interfaces.

General Terms
Design, Human Factors.

INTRODUCTION
Interactive tabletops and multi-touch surfaces are a growing area of human-computer interaction research, and recent studies have focused on the use of such devices in real world contexts including museums, galleries, and urban spaces. Indeed, informal learning environments such as museums have emerged as one area in which multi-touch interfaces can be meaningfully applied [15, 16, 17, 30]. The high-bandwidth input capabilities coupled with an inviting form factor seem ideal to facilitate the degree of social learning that is desirable in a museum context Such examinations have discovered that this context creates unique circumstances, that we call *crowd interaction* (CI) [15, 16, 17, 19, 23, 29, 30]. When designing user interfaces for crowds, one has to deal with the known "*chaos"* that arises when groups of strangers from various age groups and backgrounds spontaneously come together to "*collaborate*" (cf. Fig. 1): accidental touches are frequent [17, 36]; parallel interaction is the norm [17, 23, 29, 36]; levels of interference are high [16, 27, 29]; conflicts between participants commonly arise [16, 23, 29]; and mutual awareness amongst users regarding each other's actions and intentions cannot be considered a given [23, 29]. Additionally, UI standards for crowd interaction do not yet exist [40] and users approach gestural interfaces in a variety of different – and often conflicting – ways [16, 17, 42]. At the same time, average dwell times are low (around four minutes for a successful exhibit), making it hard to accommodate the significant training associated with teaching gestures to novice users [5, 6, 8, 11]. Overall, bringing *order* to the "chaos", while providing a UI that *everyone* can almost *instantly use*, is a significant challenge that every interface designer targeting crowd interaction must overcome.

Standard multi-touch UIs (such as the Microsoft Surface SDK [1]) provide a set of conventional widgets adapted for touch, that can be easy to learn (as they facilitate the transfer of desktop idioms, e.g. "click" → "tap") [42], and also scalable by providing functional coverage for most application scenarios. However, widgets that rely on *tapping* graphical elements on the tabletop do not perform well in chaotic crowd interaction circumstances, as they are

Figure 1. Chaos in a museum (frame from a video).

highly susceptible to interference from accidental and uncoordinated touches [17, 23, 36]. An alternative way of circumventing these issues is to design "UI-less" multi-touch applications, such as the "media-sharing"-type of application that primarily relies on the basic manipulation of graphical objects (drag/resize/rotate) [3, 4, 16, 17, 22, 29], or interaction with physical phenomena that inherently lend themselves for multi-point interaction [33, 41]. These types of applications may cope well with the chaos, but the need for scalability limits the degree to which simple physics-based approaches can be effective [42]. This motivated our efforts to go beyond "UI-less" multi-touch applications, and to push towards a fully functional UI that enables a wide spectrum of interaction semantics, while being tailored to the requirements of crowd interaction.

In this paper, we propose *FlowBlocks,* a novel multi-user, multi-touch interface that relies on *drag & drop* as the primary input primitive. Every action is triggered by dragging a *Block* and dropping it over a *Dock* (see Fig. 2). To successfully execute a function, a user has to first pick up the Block, drag it across a certain trajectory – *signaling intent to act* to other users – and drop it over the target Dock to *execute the intended action*.

Promoting Drag & Drop to the primary means of performing all system functions provides inherent advantages in the context of crowd interaction. First, it is less prone to being activated by accidental or uncoordinated touches (or "fiddling" which occurs frequently with children around a multi-touch surface), particularly when the start and end points of the drag & drop are farther apart and when other trajectory-related constraints apply. FlowBlocks provides several design parameters that a designer can use to "fortify" their UI. Secondly, *tapping* and *holding* are freed to act as initiators for feedforward mechanisms that instruct novice users on how to use the application and its UI. Thirdly, as we will demonstrate, drag & drop interaction with UI widgets introduce several design parameters that can help promote mutual awareness and conflict management between users. We have developed FlowBlocks, and established its core qualities, during a year-long process of Rapid Iterative Testing and Evaluation (RITE [24]) in one of our partner museums.

The contribution of this paper is threefold: after discussing related work, we present the FlowBlocks design space, providing details about our core principles, components, and design parameters. We then illustrate how common UI controls can be realized or replaced using FlowBlocks

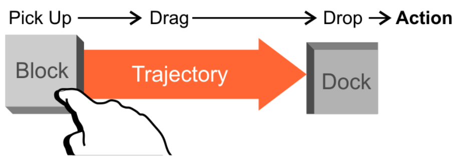

Figure 2. FlowBlocks basic principle: every action is executed by picking up a *Block*, dragging it along a certain trajectory, and dropping it over a *Dock*.

widgets, demonstrating functional coverage for the full range of application scenarios. Finally, we present the evaluation used to yield the current design of FlowBlocks, conducted while they were in use in a deployed application in one of our partner museums, and discuss our findings.

RELATED WORK

Of particular relevance to FlowBlocks is work that addresses multi-user interaction conflict management and mutual awareness, identifies problems with tap-based actions, or provides motivation for drag & drop as input primitive. Additionally, we discuss research that extends existing multi-touch UIs, UIs based on new input primitives, and tangible UIs.

Conflict Management and Mutual Awareness

A series of studies have pointed out that a variety of conflicts that commonly arise in multi-user use of interactive surfaces [7, 22, 25, 26, 29, 32, 36]. This includes conflicts through the sudden occlusion of screen real-estate or unexpected changes to the state of application [7, 29, 32], conflicts caused by accidental touch [36], or power struggles [22, 29]. Several approaches have been presented to address this issue: reducing conflicts through creating mutual awareness [7, 25]; negotiating UI access based on user identity [26, 31]; and enabling physical negotiation techniques [18, 22, 28].

FlowBlocks provides means of addressing conflict through awareness and physical negotiation: similar to the sketching gestures presented by [7]. Drag & drop provides a certain precursor to an action, particularly when performed across shared space. This approach builds on the *Privileged Object* policy [25], which suggests moving actions affecting all users to a shared space. Similar to physical objects, dragable graphical objects afford ways to utilize territoriality and negotiating for access (rather than fighting after the fact), commonly observed as natural behaviors when children interact in groups [18, 22, 28].

Morris et al. propose that social protocols are insufficient for facilitating sharing in some use scenarios, and describe a series of policies that can enforce "good behavior" among users [25]. With FlowBlocks, we follow a fundamentally different philosophy: we believe that social protocols are not inherently insufficient, but rather that UI's based on point-and-click provide insufficient cues to facilitate mutual awareness and thus underserve the needs of social protocols.

Problems with Tap-Based Actions

Existing multi-touch UIs, such as Microsoft's Surface SDK and Apple's iOS, adapt conventional UI widgets (such as buttons, drop down lists or radio buttons) to touch input, utilizing finger-tapping as the equivalent to clicking in the WIMP GUIs. While this allows a transfer of knowledge between traditional UIs and novel multi-touch systems, the lack of a hover state in touch devices eliminates helpful feedforward and feedback mechanisms [11, 40]. Further, studies observing crowd interaction report that erroneous touches lead to the frequent occurrence of false-positive invocation of actions, causing misinterpretation of UI behavior [36], and general confusion [17]. While erroneous touches can be accidental, in many cases they are intentional – caused by users who either don't realize or don't care that their actions will interfere with others. While certain mechanisms can be applied to reduce the negative effect of unintended or unnoticed touches – timeouts to prevent repeated activation [17] and confirmation dialogs [23] – these mechanisms can themselves lead to unresponsiveness, confusion [17], and tension between users [23]. Based on our own experiences in building museum exhibits, we concur with these findings, and consider standard multi-touch UIs unsuitable for crowd interaction.

Drag & Drop

Dragging is one of the most common modalities observed when users spontaneously approach tabletop UIs [16, 42]. Evidence for the success of dragging as an input modality is also provided by a series of "picture-sharing" type of tabletop applications, in which dragging responds well to interference and accidental touches [3, 16, 18, 22, 29]. For example, the "Futura" tabletop game [4] allows users can drag strategic resource tokens from a toolbar (situated at the four sides of the table) and drop them over an environment simulation in board-game like fashion (causing an effect in the outcome of an application). The authors reported that drag & drop worked well despite intense collaboration, and that users quickly learned how to place game tokens. Overall, dragging seems to be an input primitive that can be easily learned and performed by a variety of different users. This motivated us to promote dragging as the primary interaction primitive to prepare the execution of an action.

Extending Existing UIs

There is a series of work that aims at extending aspects of existing conventional UIs to multi-touch operation: the DiamondSpin Toolkit [32] provides a series of novel UI mechanisms to facilitate around-the-table interaction, such as multi-threaded input event streams and concurrent menus; Multi-touch Marking Menus aim to increase the bandwidth of touch input [21]; Attribute Gates utilize dragging trajectories across various graphical gates to adapt the object's attributes to different territories on the tabletop [35]; and Grids & Guides [12] increase the precision of multi-touch manipulations. FlowBlocks is designed for crowd interaction, and thus has less emphasis on some of these aspects, such as precision and throughput.

Graphical UI widgets that are operated through a drag-like gesture have been proposed before, albeit for other devices, motivated by different factors, and afforded with different metaphors [5, 27]. Like FlowBlocks, CrossY [5] replaces WIMP widgets with new UI controlled with a new input primitive: crossing. This is done because crossing is more efficiently and more accurately performed with a pen. Similarly, Moscovich [27] presented a set of widgets selected using sliding motions, aiming to reducing selection ambiguity on high resolution screens in which UI widgets are tightly packed. Gestures similar to "crossing" and "sliding" can be described as sub-primitives of drag & drop and have been incorporated into FlowBlocks.

Teaching Multi-Touch Gestures

One challenge for novel UIs is to be transparent to novice users [4, 5]. Consequently, there has been a series of work aimed at teaching touch and gesture-based input: to learn Marking Menu style gestures [20], OctoPocus [6] provides a dynamic guide that help users execute, learn, and remember gesture sets; ShadowGuides [11] provides in-situ guides that show a visualization of the user's fingers (feedback) alongside possible ways of continuing a gesture (feedforward). OctoPocus and ShadowGuides require the user to first understand the meaning of the visualizations they provide, thus, the process of training itself requires some degree of learning which is not practical in a CI setting where a walk-up-and-use is essential. In contrast, GesturePlay [8] provides an online learning system for teaching gestures through game-like virtual-physical widgets and positive reinforcement. Here, the mechanism for teaching gestures utilizes physical puzzles that the user can solve and in the process learn the execution of a gesture. Just-in-Time Chrome provides UI tailored to evoke various gestures [40]. It is also central to FlowBlocks to provide similar inline mechanisms that can help visitors quickly learn the operation of drag-based widgets. Elements of OctoPocus and GesturePlay have been incorporated in the design of FlowBlocks' inline-feedback and feedforward. ShadowGuide's use of the 'tap' gesture to invoke feedforward mechanisms has also been incorporated.

FLOWBLOCKS DESIGN SPACE

In FlowBlocks, the most elementary functional unit of interaction – invoking a binary command – is triggered by a *compound action* requiring the execution of two binary actions and one continuous action: acquisition of a Block (binary), dragging the Block over a Dock (continuous) and releasing the Block (binary). FlowBlocks intentionally separates the three components previously required for a binary action. In this section we will show how this separation can be beneficially used to address the challenges of crowd interaction. Based on our analysis of related work, we concentrated on designing FlowBlocks to meet these three challenges:

1. *Noise and interference*: minimize the effect of accidental touches and uncoordinated "fiddling".
2. *Mutual awareness and conflict management*: increase awareness between users regarding their intention and actions, and provide tools for negotiating conflicts.
3. *Learning how to interact*: any user must be able to use an interface almost instantly and without training.

The following subsections describe the mechanics relating to each point at the *conceptual* level of FlowBlocks. In the next section we will present *practical* guidelines for applying FlowBlocks in practice.

Dealing with Noise and Uncoordinated Interference

Generally, FlowBlocks widgets are less susceptible to accidental touches, because touches alone do not trigger an action, but only cause feedforward to appear. However, accidental touches could still coincide to form a sequence that mimics a drag operation, potentially triggering an unintended action. For this to happen, an accidental touch would have to start over the position of a Block, move across a certain trajectory, and drop the Block over the corresponding Dock. While we found this to very rarely happen in practical use, FlowBlocks provides three parameters that can further reduce the likelihood, which we will review in turn: increasing the distance between Block and Dock, requiring Dock activation, and constraining the trajectory of the drag component. These methods are not only a means of counteracting accidental touches, but can also be used to reduce the effect of uncoordinated "fiddling", which is especially prevalent among younger children.

Constraining trajectory. Figure 3 provides several examples for trajectory constraints: a) constraining movement to a path, b) using virtual barriers through which Blocks cannot pass, and c) & d) requiring certain speed of drag to allow a Block to reach the Dock. These constraints require more coordination and intent from a user in order to successfully complete an action, which reduces the action's susceptibility to noise and uncoordinated touches.

Dock activation. Docks can be configured to require explicit activation before Blocks can be dropped over their active area, which adds another layer of intent and coordination that is required to execute an action:

- *Tap-to-activate*: tapping a dock "opens" it for a certain time window.
- *Hold-to-activate*: the dock is receptive to a Block for as long as a touch is held in its active area.

While the presented mechanisms of constraining trajectories and dock activation counteract the effect of accidental and uncoordinated touches, they are also useful in the context of mutual awareness and conflict management amongst users. Several other mechanisms of FlowBlocks also help to address this design challenge.

Mutual Awareness and Conflict Management

Mutual awareness is key factor in the fluidity and naturalness of collaboration, and in allowing social protocols to facilitate interaction [13]. By being aware of other's intentions and actions, conflicts can be detected and negotiated [7, 25]. FlowBlocks' core primitive – drag & drop – consciously separates the *intent to interact* – selecting a Block and dragging it towards a Dock – from *executing the intended action* – dropping the Block over the dock (cf. Fig.3). This gives users the chance to become aware of an incoming action, decide if it conflicts with their interests, and intervene. While intervention can always be carried out through social etiquette ("Wait! Don't do that please!"), FlowBlocks also seeks to cater for more physical negotiation strategies that are commonly applied by younger children [16, 22, 28]. Consequently, each of FlowBlocks' basic components – Blocks, trajectories and Docks – affords independent ways of physical intervention. Blocks can be "claimed" and moved into a private space, or shielded from other users. Similarly, Docks can be shielded from an approaching Block, for instance by covering it with the hand, preventing the Block from being dropped. Users can also physically influence the trajectory of a drag-operation, for instance by otherwise "blocking the way", or pushing the respective hand away.

FlowBlocks draws inspiration from tangible computing [10, 14, 39, 37], particularly from the Token and Constraints (TAC) paradigm [34, 38]. Promoting virtual blocks that users can manipulate as the primary UI primitive, makes it possible to apply physical metaphors and constraints to each interface action, opening up a spectrum of design options that are not usually considered by existing, widget-based multi-touch UIs (cf. Fig. 3).

Influencing mutual awareness. FlowBlocks provide three parameters that can be adjusted when designing for awareness. First, positioning of Block and Dock are extremely important, not only because longer distances allow more time to become aware of the drag, but also because it has an impact on what portion of the interactive area has to be crossed. If drags happen in the periphery, they are less likely to cause global awareness than drags that cross shared, central space. Second, the designer can promote awareness using the same trajectory constraints we described for reducing noise and uncoordinated actions (Figure 3). Speed-dependent wrappings ensure easily observed rapid movements (c) or extra time (d). For instance, access to an important global function could be surrounded by "rubber", providing others more time for anticipation and intervention.

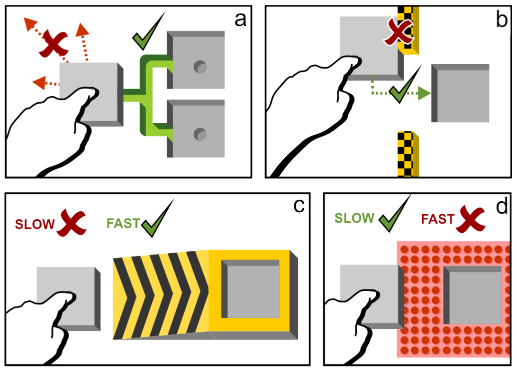

Figure 3. Constraining trajectory of drag operations: a) *rails* limit the movement of a block along a predefined path ("allow" policy); b) *barriers* define regions through which a block cannot pass ("deny" policy); c) *ramps* and d) *rubber pads* require a certain velocity of the drag operation in order to reach the target Dock.

Virtual barriers and paths create recognizable movement patterns and "bottlenecks" through which each action has to pass (a,b). These parameters can be chosen per action: actions that affect all users (e.g. reset the application) can combine any of these factors to achieve the desired global awareness, while more personal functions can omit them.

Learning from others. Existing observations have provided strong evidence that visitors learn from each other [16, 17], by copying each other's actions. This provides further motivation for increasing mutual awareness as it gives particularly novice user approaching the table, as well as younger children or elders, to learn and copy interactions observed from more advanced participants.

Managing simultaneous interactions. A special case of conflict is the simultaneous manipulation of shared resources – dubbed *global coordination* [26]. For instance, in a map application, the current view on the map is a shared resource in that changes will impact on anyone around the table, even if they are engaged in independent sub-tasks (such as looking at different parts of the map). Operations on this shared resource (e.g. jumping to another country) have to be regulated in that only one person can manipulate the resource at a time. In crowd interaction, the common case is that a single person triggers such an action, while others are interrupted in their task, and potentially confused about why this action is happening. FlowBlocks can be used as described previously to create awareness and enable anticipation of global events. Additionally, the Block-Dock metaphor inherently conveys what actions can be executed simultaneously. If a Dock is occupied, it can be "locked", causing any attempt to drop another Block into it to "bounce off", until the associated action is completed (e.g. the view has zoomed to a new country on the map). It is also possible to configure Docks to remain open, so that the existing action can be interrupted by removing the occupying Block, or by dropping a new Block onto the Dock, causing the previous one to "fall out". While this mutual locking is possible with existing multi-touch widget through "graying out" UI elements that are currently inaccessible, FlowBlocks inherently convey and reflect the state of execution, and the availability of functions.

Learning

Due to the lack of UI standards for crowd interaction [40] and a variety of different ways in which users approach gestural interfaces [16, 17, 42], any crowd UI has to clearly convey its meaning to inexperienced users. This aspect is particularly emphasized in a museum context, as dwell times are usually short, and as a consequence, the learning of the interface cannot take up more than a few seconds.

In conventional GUIs, the pointer's hover state can be used to instruct novice users how to interact – such as to show tooltips when the mouse pointer is dwelling over a control. Preview states, however, are sorely missed in a

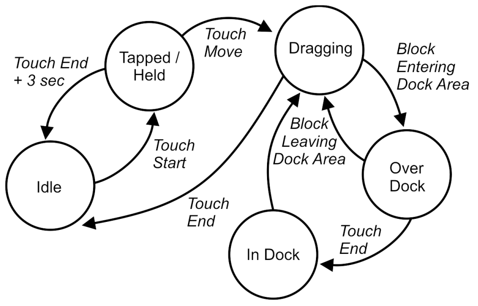

Figure 4. FlowBlocks' *Block* state diagram.

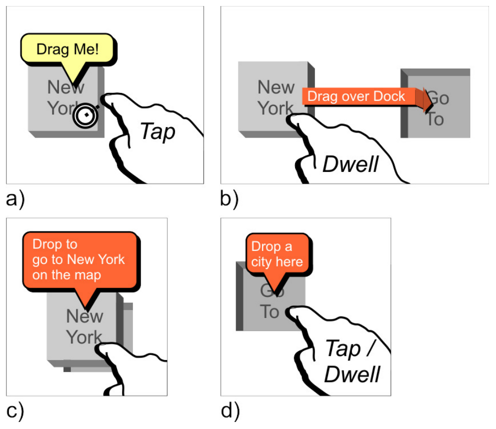

Figure 5. Feedforward help: a) prompting to drag; b) showing directional guides leading towards compatible Docks and c) preview before the execution of an action. Managing screen clutter: d) using relative guides, e) weighing guide strength by distance to Dock.

straightforward port of WIMP UI to touch environments [11, 40]. FlowBlocks introduces a novel state model for its widgets that adds two additional states to existing state models for touch-sensitive input devices [9]. Figure 4 shows the state-transition diagram for a Block. As soon as a Block is touched, it enters a *Tapped / Held* state. This state is also retained for at least 3 seconds in case the user immediately retracts the touch – ensuring that a 'tap' gesture will also activate this state for a short duration. A second state *Over Dock* is entered as soon as a Block is moved over a dock (but not released yet). The remaining states correspond to the traditional three-state model of input [9]: *Idle ~ State 0* (out of range), *Dragging ~ State 1* (tracking) and *In Dock ~ State 2* (state 2 refers to the activation of the input device, in our case, a Block occupying a Dock).

Preview states can be used to teach both mode of operation, and mechanics of the application. Based on our state model, we can provide instructions at three moments where we are able to infer possible user confusion. First, we need to anticipate any graphical element to be tapped, including both Blocks and Docks, as tapping is the most consistently observed way users spontaneously interact with graphical elements [16, 36]. While in the state model *Tapped / Held* does not cause an action in FlowBlocks, we can utilize this state to convey the proper mode of operation to the user.

For Docks, we can provide appropriate feedback (cf. Fig. 5d). For Blocks, we can instruct the user to drag, for instance, via a tooltip (cf. Fig. 5a). In case of a Block, instructions can persist throughout dragging and provide updated instruction and feedforward to help the user with the proper execution of an action, such as guided by arrows (Fig. 5b). Generally, all existing feedforward techniques can be applied, such as OctoPocus [6] or ShadowGuides [11]. The second moment of potential confusion within our state model regards the mechanics of the application. The *Over Dock* state can be used to provide a preview of what *would* happen if the currently dragged Block was dropped over a Dock (cf. Fig. 5c). This is equivalent to a traditional tooltip.

Having designed FlowBlocks' many parameters to overcome specific challenges inherent in crowd interaction, we now demonstrate their functional completeness. To do this, we designed FlowBlocks-based widgets that mimic the logical function of traditional WIMP UI elements.

FLOWBLOCKS WIDGETS

A significant challenge in the construction of UI elements based on a new physical primitive is the need to ensure functional coverage for application development. Previous efforts have accomplished this by redesigning WIMP UI widgets for use with their new primitives [5]. Our intention is not to mimic earlier controls, but rather to use them as a spanning set of logical actions required to create useful applications. Based on common GUI toolkits, we have identified 4 logical types of components that are required to cover most interaction semantics: *Binary Triggers, Selections, Scalar Controls,* and *Menus*. We omit logical compound components as long as they can be constructed using the five types of components we present (e.g. a File Open Dialog is a compound widgets, consisting of a scrollable container and binary controls).

Figure 6 shows each of the five classes of controls that can be supported by FlowBlocks. We provide examples for three different *binary controls*: slider-trigger controls (a), similar to the "Unlock"-slider in iOS; (b) "url" like navigation based on a "Navigation"-Dock and several "URL"-Blocks; (c) an arrangement of two slider-triggers to peek into and open a photo album – combining FlowBlocks widgets in this way might lend itself to playfully teaching multi-point gestures to novice users, similar to [8]. FlowBlocks can support many types of *selectors* through the assembly of Blocks and Docks, as in examples (d) and (e). While Docks usually are limited to one occupant, they can also be considered as containers in which several elements can be placed (e). *Sliders* can be constructed by constraining a handle-Block within a rectangular area using a barrier, and filling the area seamlessly with as many docks as there are steps (f, g, h). Further mechanisms can be put into place to prevent the accidental adjustment of a slider: "gear-box-style" rails require the user to first lift the

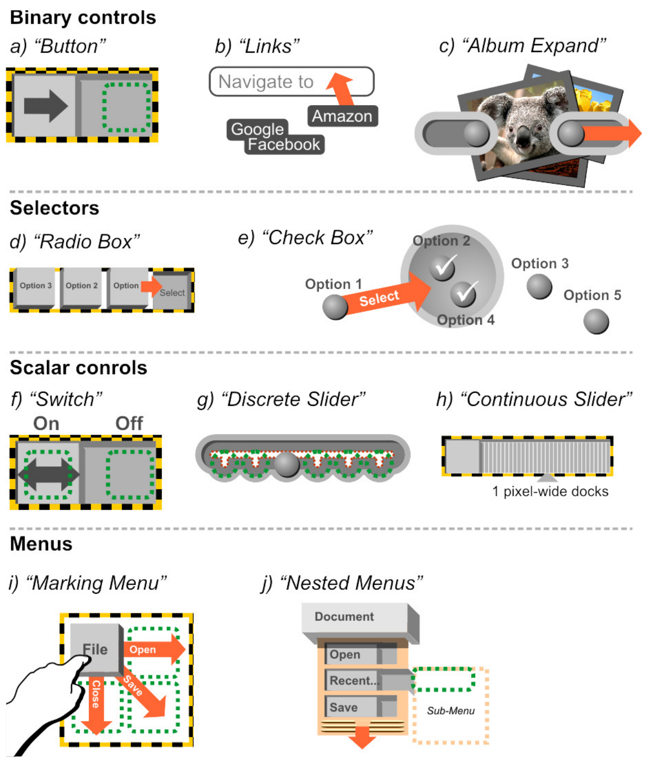

Figure 6. Examples for FlowBlocks versions of common types of UI widget:binary controls, selectors, scalar controls and menus; Legend: 🟥 = rail, 🟩 = invisible Dock, ▨ = invisible barrier.

handle before it can be horizontally adjusted (g). FlowBlocks also affords various menu styles (i, j). Example (i) shows a menu in the form of a single Block. Once the user touches the Block, arrows point into three distinct directions, and activates invisible docks, as well as an invisible barrier that keeps the Block within the allowed area. If the user now drags the Block in a direction beyond the bounds of the barrier, it will automatically drop over one of the invisible Docks. Thus FlowBlocks are used to implement the scale-independence and free-movement of a Marking Menu [20]. These menus illustrate two more concepts that FlowBlocks can support: dynamically showing and hiding its constituent elements, as well as linking different states from our model to create more complex UI behaviors.

Having demonstrated that FlowBlocks are logically complete across the functions of the WIMP GUI, we now address their design and integration into an application, as well as the iterative design process that led to the development of many of the design parameters we have discussed.

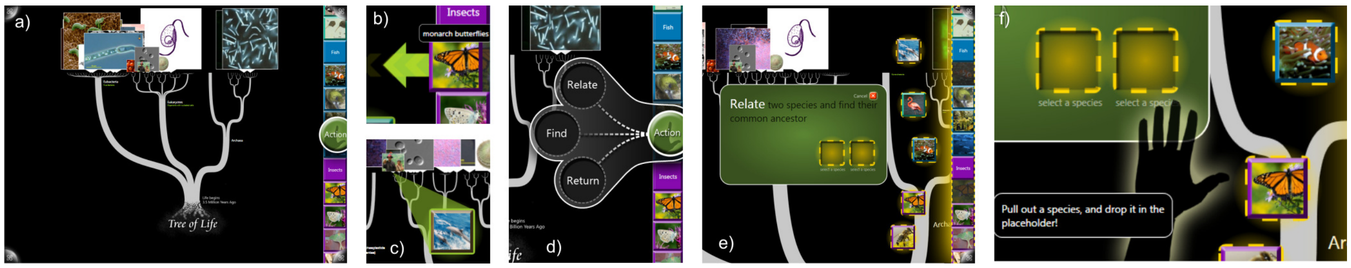

Figure 7. The DeepTree exhibit (a) and FlowBlocks UI (b, c, d, e, f).

APPLICATION AND EVALUATION

FlowBlocks is the outcome of a yearlong iterative design process in which we implemented a Microsoft Surface application, *DeepTree*, as part of a 3-year NSF *Informal Science Education* project. We directly observed over 280 visitors at our DeepTree testing exhibit using talk-aloud protocols. The DeepTree is an interactive and multi-touch visual application that allows visitors to explore the evolutionary relationship and historical time of divergence of all life on earth. This relationship is visualized in the form of a large phylogenetic tree based on data from the Tree of Life Web Project [2] with over 70.000 species, and 20.000 internal nodes. Figure 7a shows DeepTree application and its interface.

The main area shows the Tree of Life, as well as *images* that mark the location of representative species in the tree (in the illustration, the root of the tree is visible, showing the three domains Bacteria, Eukaryotes and Archaea). Users can browse the tree of life with drag-based manipulation, dragging the tree downward to zoom-in towards the leaves, or upward towards the root to zoom-out.

At the right side of the screen, a scrolling image reel of 200 selected species-Blocks are shown. Visitors can scroll through the image reel by standard flicking techniques, or pull any species from the reel and move it above the tree (cf. Fig. 7b). Here, the Blocks gain a "chord" that visually links the block with the location of the species in the tree, to aid navigation and orientation (cf. Fig. 7c). Users can also perform *Find* and *Relate* queries using our Flow Blocks UI. An "Action" marking menu on the right side provides access to *Find*, *Relate* and *Return* (cf. Fig. 7d). Once dragged into a direction and dropped, the action Block morphs into a dialog which occupies the central area of the display (c.f Fig. e). The dialog presents one (*Find*) or two (*Relate*) Docks, respectively. The user executes an action by dragging species-Blocks into the Docks (cf. Fig. 7f). When a species-Block is dropped into a Dock, the tree seamlessly zooms and pans to show an appropriate view (similar to a map flying to a new city). For *relate* queries, this is a view of the most recent common ancestor (the point in the tree were both species separated), as well as visualizations of traits shared by the species which were dropped-in to the Docks. The *Return* dialog contains a trigger-slider (Fig. 6a), which flies the user back to the root of the tree.

The aim of the exhibit was to allow visitors to explore evolutionary concepts and to casually browse the tree for any of the 70,000 species. The comparisons and searches provide a focused activity, and a natural end point, which is recommended in public venues with large crowds [36].

Method

As part of our iterative design process, we used the RITE method (Rapid Iterative Testing and Evaluation [24]) throughout the year, testing the various stages of development and alternative versions of FlowBlocks inside one of the galleries at our partner museum. This method was well suited for our project for reasons of external validity: we wished to carry out our evaluations in the native setting where our exhibit will be installed when it is finished, and with actual visitors that spontaneously interact with our system. Our partner museum has over 170,000 visitors per year from diverse backgrounds. Observations were carried out during weekdays – typically attracting school groups and summer camps – as well as on weekends, when many families visit the museum.

RITE entails expert observations to drive quick design iterations. Both video footage and field notes were collected. Institutional Review Board permission was granted to gather video recordings of visitors without prior written consent, however, a video-taping notice had to be present during observations, and collected footage could only be used for internal analysis. Additionally, an observer was physically present, shadowing the museum visitors and taking notes from a discreet distance. After visitors finished interacting with the DeepTree they were approached by the observer and given the opportunity to provide verbal feedback, which could inform our design iterations. To reduce bias, an independent researcher who was not involved in the development and design of FlowBlocks carried out all observations. The criteria used to evaluate each RITE iteration were based on the three pillars of the presented design space:

1. Create *mutual awareness*, and enable *conflict negotiation*, be it verbal or physical.
2. Reduce (or eliminate) invocations of actions through accidental and uncoordinated touches (*noise & interference*).
3. Visitors should learn to use FlowBlocks during the (typically short) time they interact with the system (*learning to interact*).

New design iterations were begun as soon as it became clear that the design goal was not yet met, or when bugs prevented us from carrying out meaningful observations. Deployments between iterations varied in length and number of users, but were typically 1 week and approximately 20-40 observed users. In total, 10 iterations were conducted over 15 months, with over 280 visitors being observed. With this approach, our designs have been tested beyond most of the reported "*in the wild*" work [16, 23] in that we did not stop at one or two rounds of evaluation. Instead, our designs have been both designed and evaluated in the wild.

Observations & Findings

The *FlowBlocks Design Space* and *Widgets*, and *Application* sections describe the finished product of the RITE process, as well as several design parameters which, while clearly useful in some contexts, did not suit the particular design of the DeepTree exhibit. We now discuss observations gathered throughout the iterative design process, which will be of benefit to researchers and designers seeking to extend and apply FlowBlocks, and to those designing UI for crowd interaction in general.

Everyone can Learn to Drag

We found that nearly all users approach FlowBlocks in one of two ways: some instantly start dragging Blocks around and dropping them into Docks. Particularly children seem to require no instruction as to how FlowBlocks work, which indicates that they could infer the proper use from the visual design of Blocks and Docks shown in Figure 7e. These visuals were the 11th version tested. The second category of user starts by tapping Blocks or Docks. While at this point, feedforward can help the transition from tapping to dragging, we went through several iterations for this transition to happen quickly, and for all age groups. Initially, we used arrow guides that marked the potential routes from the dragged Blocks to the Docks, similar to those described in Octopocus [6]. The arrows persisted through the act of dragging and updated dynamically. While this effectively encouraged hesitant visitors drag, for kids, who readily drag as many species over the tree as possible, this technique lead to significant screen clutter, as many guides cross large parts of the display space.

After several iterations, the final feedforward consisted of four mechanisms. First, if species Blocks within the image reel are tapped, for a brief moment the Block is slightly nudged outwards, and an animated arrow appears (Fig. 7b). This arrow is relatively small, and disappears as soon as the Block is dragged. This was effective in initiating the transition to drag, while not causing screen clutter during dragging, even when multiple elements where pulled out at the same time. Secondly, we used an animated visual effect of an ant-trail (cf. Fig. 7f) that surrounded both Docks and Blocks – to convey their functional dependency. This effect was only visible when an action dialog was in the center of the display (after being selected from the action menu). Thirdly, when tapping a dock, an animated hand shows up, performing a drag operation from the image reel to the Docks (cf. Fig. 7f). The described feedforward was effective for transitioning most visitors to dragging, but particularly seniors still showed hesitance and signs of confusion, even when all three mechanisms where active. Thus, we added a tooltip which was displayed when tapping either the Blocks or the Docks. The tooltip gives instruction in written form, and we found indicators in form of utterances during use that this textual information was effective in instructing adults and seniors, while not hindering kids (who do not read).

After these modifications, FlowBlocks were consistently learned within the first few seconds of interaction. All groups approaching the table, including school groups and seniors, started carrying out actions almost immediately. Considering that UI standards for crowd interaction do not yet exist [40] and that user intuition regarding the "proper" use of such UIs widely differs [16, 17, 42], FlowBlock is effective in teaching a consistent operation of UI widget to a wide range of users. We found less evidence that users noticed the second type of feedforward (shown when a Block is dragged over a Dock). Users seemed willing to try out an action without precise foreknowledge of the result. This may be less true in settings where errors are costlier.

Little Effect of Noise and Fiddling

Our initial prototype of the exhibit was based on tap-activated species-markers. Concurring with existing findings [17, 23, 36], we quickly disqualified this mode of invoking commands for our context, as accidental and random jumps prevented constructive collaboration, particularly in large groups. After introducing FlowBlocks, we did not observe any instances in which accidental touches triggered one of our actions, or in which users seemed to be confused about why a certain action happened. Furthermore, active "fiddling" (children quickly wiping their fingers across the species-Blocks) very rarely caused issues other than multiple feedforward guides cluttering the screen, which led to the described feedforward designs. Overall, we were satisfied with the effectiveness of FlowBlocks regarding both accidental as well as uncoordinated touches, as we generally observed constructive execution of actions despite intense interference.

Drag & Drop is Hard to Ignore and Easy to Prevent

We frequently observed occurrences in which users either verbally or physically reacted to another user's action *prior* to a Block being dropped. We infer from this that users were generally aware of each other's actions, particularly of those actions that we intended to require consent. This was apparent after some initial play and learning about FlowBlocks, children would transition to more constructive interaction with the system, and actively prevent other participants from executing interfering actions. Our observation concurred with existing work in that physical strategies are commonly deployed by younger children, such as blocking one another's hands, pushing each other away, or covering Blocks or Docks to claim "territory" [16, 22].

Many Solutions to Affect Awareness and Collaboration

Throughout our iterations, we tested various parameters of FlowBlocks and their effect on mutual awareness. In an early version (Fig. 8a&b), both the Docks for *Find* and *Relate*, as well as the reset slider were visible at all times, and placed in the top portion of the display. While this created functional overview, we did notice occasions on which visitors seemed surprised and unaware of other visitors triggering a *Find*, *Relate*, or *Return (*then called *Reset*), and at times uncertain about which function (*Find* or *Relate*) was currently executing. This was particularly apparent in a version in which Find and Relate Docks were spatially separated (cf. Fig. b). Several iterations addressed this issue. In one design, we tried to first only show the *Find* Dock (then "Jump To", cf. Fig. 8c), then, after selecting and flying to a species, show a *Relate* Dock (then called "Compare With", cf. Fig. 8d). Consequently, after selecting a species for the *Find*-action, the species is now the starting point for a *Relate*-query. In contrast to the previous versions (Fig. 8a & b), this design made sure that

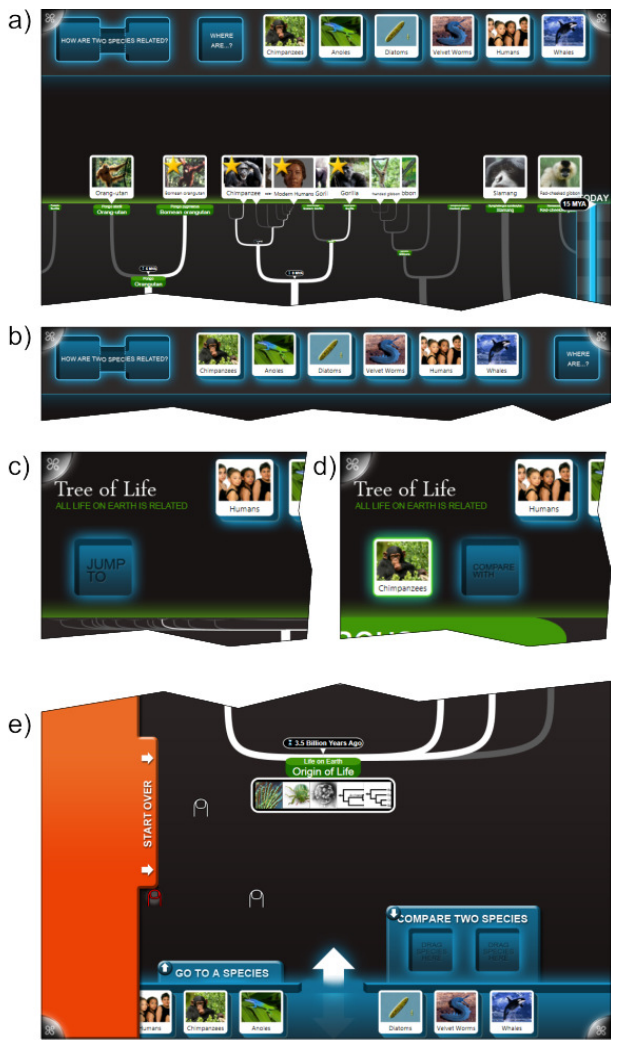

Figure 8. Selection of discarded design iterations of the DeepTree UI.

only the active Docks of *one* function – Find *or* Relate – are visible at a time, and that both actions are presented in the same area on the screen. While this design eliminated the described awareness issues, the mechanism of a Dock changing its function *after* it has been originally occupied with a Block appeared to be a separate source of confusion. Visitors could not anticipate this conditional change when making their initial choice, nor did our design visually communicate this change of functional meaning (the Dock's label is occluded once occupied by a Block). For our exhibit, we dismissed conditionally changing Docks, however, they might be suitable for other scenarios.

An alternative way of addressing the awareness issues observed during earlier iterations (Fig. 8a&b), was to utilize several of the described mechanisms to increase the precursor necessary for each action (Fig. 8e). We place virtual walls so that the species Blocks could only be dragged out of a small bottleneck at the center. The species docks for *Find* and *Relate* are hidden in "drawers", and first have to be pulled out before being accessible (a form of Dock-activation). Additionally, we made the *Reset* (or *Return*) action accessible via a slider, which when pulled out starts to cover the whole screen in a red "curtain". The slider has to be pulled across the whole screen and released in order for the Reset action to trigger.

In a study within one of our iterations in which we compared both interfaces (Fig 8a & e), we found the second version reduced occurrences of confusion, by introducing new constraints that made it harder for a drag to go unnoticed. However, it was also less popular for adults, as the effort of invoking an action was relatively high. In contrast, kids loved this version (cf. "Constraints vs. Play"). In our final design, we utilized the described marking menu and a positioning of the action dialogs in the center of the display (Fig. 7b). This design had three benefits: the marking menu added another precursor to an action; positioning the the action's Docks into shared space (as advocated by [25]) brought the Docks of an incoming action to everyone's attention; and showing only Docks of a single function at a time reduced the confusion observed with earlier designs (Fig. 8a & b). The final UI version balanced suitability for a large age range with good awareness characteristics.

Constraints vs. Play

By providing a set of loose movable tokens, FlowBlocks inherently lend themselves to playful interaction. We have observed various forms of play with the DeepTree UI. FlowBlocks originally implemented inertia, so the species Blocks could be flicked around the table, causing particularly younger kids to "play hockey" (as one of our younger visitors called it). Another popular game among kids was to drag as many species blocks onto the tree as possible. While this form of play seemed enjoyable, it distracted from "deeper" engagement, particularly for younger audiences.

As the DeepTree is focused on learning, we experimented with constraints to "enforce order", such as disabling inertia, introducing the bottleneck (Fig. 8c), and forcing a one-at-a-time order for pulling out species Blocks. While these modifications reduced levels of play, the overuse of such constraints resulted in slow and awkward interaction experiences for adults. Likewise, we observed instances in which the *presence* of constraints encouraged an "obstacle course" style of play (e.g. routing a Block through the mentioned bottleneck). Generally, we feel that certain levels of play are inherent to FlowBlocks, which has to be considered by the UI designer. In our final design, we tried to strike a balance of allowing certain levels of play while minimizing their impact on mature participants (see "Managing Screen Clutter"). This can enable little kids to "participate" alongside more focused and deeper interaction.

Managing Screen Clutter

One issue we have identified is that Blocks scattering around the Display during usage (due to fiddling or directed use) can cause significant screen clutter. We have found two mechanisms that deal with this issue: first, Blocks that are not dropped over a Dock can snap back to their original location; secondly, Blocks can be configured to disappear if they are not actively touched. In our design, the Blocks dragged onto the tree started to slowly fade out when not touched, and eventually move back to the reel. This was successful in reducing clutter.

Sequential vs. Parallel Interaction

As described in the design space, FlowBlocks can be configured to enable parallel interaction or to enforce sequential execution of actions. While one of the obvious strengths of multi-touch technology is its ability to allow multiple users to interact at the same time, we have found that sequential interaction also has merits. For learning, it is preferable for visitors working together rather than working side-by-side. At the macro level, the FlowBlocks UI enforces a parallel execution of actions in order to create a shared context for all participants. However, at the micro-level, phases of negotiation and simultaneous interaction were frequently observed, both verbally, as well as facilitated through moving species Blocks close to the Docks to signal individual preference for which parameter to chose next. We frequently observed turn-based interaction, and at occasion a modality of actor and observer(s). However, both modes of operating the DeepTree were successful in engaging participants in a shared learning activity.

In summary, we found that FlowBlocks, in their current design, and with various parameters, can support a variety of different designs that allow UI designers to actively influence how certain actions are perceived, and enable the creation of successful, crowd-friendly UI.

CONCLUSION

Although the challenges of crowd interaction are widely acknowledged, there are no general UI frameworks or design guidelines that can inform the concrete design of crowd UIs across a variety of application scenarios. Our work starts to fill this gap by providing a general UI, based on the drag-and-drop primitive that is custom-tailored to the challenges of crowd interaction. Our year-long iterative design and evaluation has established the founding qualities of FlowBlocks – enabling crowd interaction in the face of chaos: its robustness regarding accidental touches and interference, its ease of learning and use for a wide spectrum of users, and its capability to provide application designers with a variety of options to influence mutual awareness, and facilitation of conflict management. We have also shown that FlowBlocks provide common interaction primitives that afford the functional scalability of common UI for multi-touch interaction.

While our insights have confirmed that drag-based widgets work well for crowd interaction for the DeepTree museum exhibit, further studies are required to understand how best FlowBlocks can be extended to support wider application scenarios. Theoretically, FlowBlocks can support "desktop-grade" functionality in crowd interaction settings, given their coverage of existing UI primitives. We certainly do not envision to port complex applications, such as photo editing software, to crowd interaction. We do believe, however, that there are many meaningful application scenarios that benefit from more complex interaction semantics compared to "picture-sharing" type of interaction commonly found on surface computer. The DeepTree exhibit serves as an example of more complex interaction flow.

In museum and public contexts, UI composition has significant impact on whether visitors have an enjoyable social experience in which they acquire knowledge, or leave frustrated and confused. Every effort to optimize the UI can thus potentially have significant impact on a wide range of users. DeepTree will soon be deployed in five different museums, with an estimated 4.5 million total visitors in the next year. FlowBlocks will serve to enable visitors to enjoy a meaningful and playful learning experience.

ACKNOWLEDGEMENTS

We thank the Harvard Museum of Natural History, and its Director of Exhibitions, Janis Sacco, for allowing us to conduct this research in their galleries. We also thank Kay Hofmeester and Paul Hoover, formerly of the Microsoft Surface team, for their discussion on drag & drop-based widgets. We thank the National Science Foundation for their support of this project (DRL-1010889). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

1. Microsoft Surface. http://www.microsoft.com/surface/
2. http://www.tolweb.org
3. http://www.ideum.com/interactive-exhibits/
4. Antle A. Futura: design for collaborative learning and game play on a multi-touch digital tabletop. *TEI'11*, 93-100.
5. Apitz, G. et al. CrossY: a crossing-based drawing application.. *UIST '04*,3-12.
6. Bau, O. et al. OctoPocus: a dynamic guide for learning gesture-based command sets. *UIST '08*, 37-46.
7. Belatar, M. et al. Sketched menus and iconic gestures, techniques designed in the context of shareable interfaces. *ITS '10*, 143 – 146.
8. Bragdon, A. et al. Gesture play: motivating online gesture learning with fun, positive reinforcement and physical metaphors. *ITS '10*, 39-48.
9. Buxton, W. A three-state model of graphical input. *INTERACT '90*, 449-456.
10. Cohen, J. et al. Logjam: a tangible multi-person interface for video logging. In Proc. CHI '99, 128-135.
11. Freeman, D. et al. 2009. ShadowGuides: visualizations for in-situ learning of multi-touch and whole-hand gestures. *ITS '09*, 165-172.
12. Frisch, M. Et al. Grids & guides: multi-touch layout and alignment tools. *CHI '11*, 1615-1618.
13. Gutwin, C. et al. A descriptive framework of workspace awareness for real-time groupware. *CSCW 2002, 411 – 446.*
14. Hartmann, B. Pictionaire: supporting collab*orative design work by integrating physical and digital* artifacts. *CSCW '10*, 421-424.
15. Hinrichs, U. et al. Emdialog: Bringing information visualization into the museum. *IEEE ToV&CG*, 14(6):1181–1188, 2008.
16. Hinrichs, U. et al. Gestures in the wild: Studying multi-touch gesture sequences on interactive tabletop exhibits. *CHI 2011*, 3023-3032.
17. Hornecker, E. I dont understand it either, but it is cool-visitor interactions with a multi-touch table in a museum. *TABLETOP 2008*, 113–120.
18. Hornecker, E. Interactions around a contextually embedded system. *TEI '10,* 169 – 176.
19. Jacucci, G. et al. Worlds of information: designing for engagement at a public multi-touch display. *CHI '10*, 2267–2276.
20. Kurtenbach, G. et al. 1994. User learning and performance with marking menus. CHI'94, 258-264.
21. Lepinski, G.J. et al. The Design and Evaluation of Multitouch Marking Menus. *CHI '10*, 2233 – 2242.
22. Marshall, P. et al. Fighting for control: children's embodied interactions when using physical and digital representations. In *CHI '09*, 2149 – 2152.
23. Marshall, P. et al. Rethinking'multi-user': an in-the-wild study of how groups approach a walk-up-and-use tabletop interface. In CHI '11, 3033--3042 2011.
24. Medlock, M. et al. The Rapid Iterative Test and Evaluation Method: Better Products in Less Time. *Cost Justifying Usability, An Update for the Internet Age.* Bias, R. and Mayhew, D. (eds). Boston, Morgan Kaufmann, 2005.
25. Morris, M.R. et al. Beyond social protocols: Multi-user coordination policies for co-located groupware. *CSCW'04* 262 – 265.
26. Morris, M.R. et al. Cooperative gestures: multi-user gestural interactions for co-located groupware. *CHI'06* 1201 – 1210.
27. Moscovich, T. 2009. Contact area interaction with sliding widgets. *UIST '09*, 13-22.
28. Olson, I.C. et al. "It's just a toolbar!" Using Tangibles to Help Children Manage Conflict Around a Multi-Touch Tabletop. *TEI'11*, 29-36.
29. Peltonen, P. et al. It's mine, don't touch!: interactions at a large multi-touch display in a city centre. *CHI'08* 1285–1294.
30. Schmidt, H. et al. memory [en] codebuilding a collective memory within a tabletop installation. *Proc. CAe*, Eurographics Association, 135–142, 2007.
31. Schmidt, D. et al. IdLenses: Dynamic personal areas on shared surfaces. *ITS 2010*, 131-134.
32. Shen, C. et al. DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction. *CHI'04*, 167-174.
33. Scott, S. et al. 2009. Social immersive media: pursuing best practices for multi-user interactive camera/projector exhibits. *CHI '09*, 1447-1456.
34. Shaer, O. et al. The TAC Paradigm: Specifying Tangible User Interfaces. PUC, vol. 8, no. 5, pp. 359-369, Sept. 2004.
35. Sulaiman, N. S. et al. Attribute Gates. *UIST 2008*, 57 – 66.
36. Ryall, K. et al. Experiences with and observations of direct-touch tabletops. *Tabletop*, 89-96.
37. Ullmer, B., Ishii, H., and Glas, D. mediaBlocks: physical containers, transports, and controls for online media. In Proc. SIGGRAPH '98, 379-386.
38. Ullmer, B., et al. Token+Constraint Systems for Tangible Interaction with Digital Information. ToCHI 2005, pp. 81-118.
39. Weiss, M. et al. 2009. SLAP widgets: bridging the gap between virtual and physical controls on tabletops. *CHI '09*, 481-490.
40. Wigdor, D., Wixon, D. *Brave NUI World | Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann, 2010.
41. Wilson, A. et al. 2008. Bringing physics to the surface. *UIST '08*, 67-76.
42. Wobbrock, J. et al. 2009. User-defined gestures for surface computing. *CHI '09,* 1083-1092.