# Astral: Prototyping Mobile and Smart Object Interactive Behaviours Using Familiar Applications

**David Ledo[1], Jo Vermeulen[2], Sheelagh Carpendale[1], Saul Greenberg[1], Lora Oehlberg[1], Sebastian Boring[3]**

[1] Department of Computer Science, University of Calgary, Canada · {first.last}@ucalgary.ca

[2] Department of Computer Science, Aarhus University, Denmark · jo.vermeulen@cs.au.dk

[3] Department of Computer Science, University of Copenhagen, Denmark · sebastian.boring@di.ku.dk

## ABSTRACT

*Astral* is a prototyping tool for authoring mobile and smart object interactive behaviours. It mirrors selected display contents of desktop applications onto mobile devices (smartphones and smartwatches), and streams/remaps mobile sensor data to desktop input events (mouse or keyboard) to manipulate selected desktop contents. This allows designers to use familiar desktop applications (e.g. PowerPoint, AfterEffects) to prototype rich interactive behaviours. Astral *combines and integrates* display mirroring, sensor streaming and input remapping, where designers can exploit familiar desktop applications to prototype, explore and fine-tune dynamic interactive behaviours. With Astral, designers can visually author rules to test real-time behaviours *while* interactions take place, as well as *after* the interaction has occurred. We demonstrate Astral's applicability, workflow and expressiveness within the interaction design process through both new examples and replication of prior approaches that illustrate how various familiar desktop applications are leveraged and repurposed.

### Author Keywords

Prototyping; design tool; interactive behaviour; smart objects; mobile interfaces.

### CCS Concepts

**• Human-centered computing → Interface design prototyping; User interface toolkits**

## INTRODUCTION

Smart interactive devices such as mobile devices, wearables and smart objects vary widely in input and output, physical form, and development platforms. When prototyping interactive behaviors for these devices, designers are faced with two options. The first option is to build prototypes directly on the target device or platform. This involves programming and, in some cases, circuit building or soldering (when prototyping in physical computing). As a result, implementation details consume the majority of designers' time and resources (e.g., code setup, learning the platform, programming basic visuals), instead of important early-stage design decisions such as exploring alternative solutions or defining elements of animation and interaction [13]. The alternative option is to explore a device's interaction via desktop-based prototyping tools (e.g. InVision). However, such prototyping tools often specialize in specific tasks (e.g. wireframing, standard UI widgets, creating animations as videos) without room for integrating different workflows [39]. Furthermore, while some tools support deploying prototypes on mobile devices, they often do not support live sensor input other than touch events on simple trigger action states. This limits designers' ability to envision *how* the interaction might play out on the target device [25]. Exploring and fine-tuning responsive behaviours is crucial for interactions that (1) go beyond standard UI patterns on mobile platforms; (2) involve the target device's physical form or possible physical actions (e.g. shaking the device); and (3) contain nuanced continuous animations that play out *as the inputs are happening* (e.g. slide to unlock). In these situations, designers should be able to

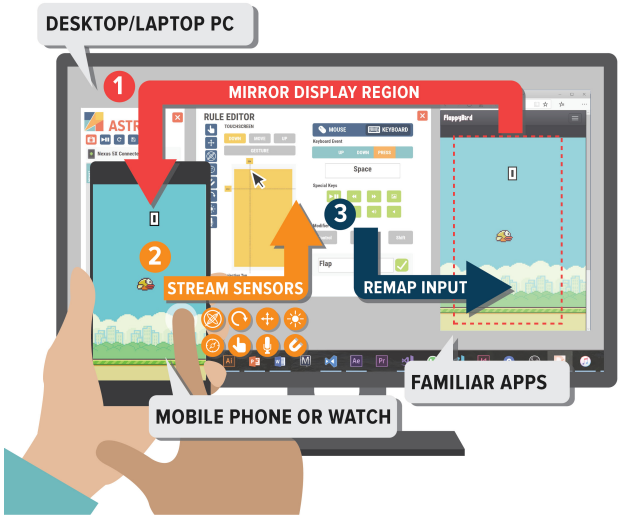


**Figure 1. Astral allows designers to prototype interactive behaviours by (1) mirroring contents of a desktop region to a mobile device, (2) streaming mobile sensor data to the desktop, and (3) remapping the sensor data into desktop input (e.g. mouse and keyboard events) on a designer-chosen desktop application.**



*DIS '19,* June 23–28, 2019, San Diego, CA, USA

ACM ISBN 978-1-4503-5850-7/19/06 $15.00
https://doi.org/10.1145/3322276.3322329

explore the dynamic interplay between inputs and outputs *as* interaction happens. We aim to empower designers to envision these responsive behaviours in their prototypes, a process that is currently not facilitated by existing tools in research and industry.

## ASTRAL

We introduce Astral (Figure 1 & video figure), a prototyping tool that uses mirroring, streaming and input remapping via simple rules to enable designers to author, test, and fine-tune interactive behaviour in mobile devices using familiar applications. We achieve this through the following three steps:

***1. Mirroring Desktop Visual Content to a Mobile Display.*** Astral consists of a desktop client and a mobile client application. As shown in Figure 1.1, the designer selects a region (here a web browser running Flappy Bird) to mirror it to a connected mobile (phone or watch) display. The mobile display is then updated live as the desktop display refreshes.

***2. Streaming Mobile Sensor Data to Provide Input.*** As shown in Figure 1.2, a designer can select data from multiple types of sensors, such as touch, acceleration, ambient light or the microphone. In this case, the designer enables the touch sensor to be streamed back to the Astral desktop client. Every time the designer now taps the mobile display, the touch location is visualized on the Astral desktop client.

***3. Interactivity via Input Remapping.*** As shown in Figure 1.3, the designer can provide interactivity to the mirrored display by remapping the mobile sensor data to desktop keyboard or mouse inputs. Here, by remapping a mobile touch down event to a desktop spacebar keypress, the bird in the mirrored web browser can flap its wings. The designer can visually select the entire screen of the phone to sense touch events. Designers now have a greater range of expressiveness: they can prototype and fine-tune interactive behaviours, such as continuous and discrete actions, apply interactive animation easing functions, or emulate states.

While we primarily focus on mobile interactions, Astral's building blocks and workflow also translate to prototyping smart object behaviours. Instead of challenging designers to solder, create custom circuitry and program electronics [5], our mobile focus by extension allows designers to place mobile devices into a fabricated shell and repurpose their inputs and outputs (e.g. as shown in Soul–Body Prototyping [32], Nintendo Labo [68], Sauron [56], AMI [57]) and continue working with familiar desktop applications.

### Benefits and Contributions

Astral is informed directly by (1) prior formative studies [39, 49, 63, 64]; (2) state of the art tools in research and industry; and (3) our experiences talking to designers, teaching interaction design to designers and computer scientists, and creating prototyping tools and toolkits for behaviour authoring. Astral makes the following contributions:

**1. Enabling Designers to Leverage and Repurpose Familiar Applications to Prototype Interactive Behaviours.** Astral *combines and integrates* display mirroring, sensor streaming and input remapping, where designers can exploit desktop applications (e.g. PowerPoint, AfterEffects, Chrome) [39, 63, 64] in a novel manner for prototyping interactive behaviours. Because any desktop application can be mirrored via Astral, any sensor data can be remapped to the mouse and keyboard events understood by that desktop application. Astral facilitates a tight design-test-refine cycle on target devices without needing to write code through encapsulation of sensor data and mappings into rules, and immediate interactive preview. Astral is designed to support various interaction design practices (*getting the right design* [16]), including: video-based prototyping, prototyping multiple alternatives, iterative prototyping, and smart object prototyping (by placing mobile devices into physical forms [32, 56, 57, 68]).

**2. Allowing Designers to Visually Author Dynamic Interactions Based on Continuous Sensor Data.** The common state model approach (e.g. d.tools [23], Adobe XD, inVision) does not lend itself well to dynamic visuals and fine-tuned temporal aspects of user interface behaviour. By working with and visualizing continuous streams of sensor data, and mapping (and easing) these to continuous outputs, our work enables authoring "*user-in-the-loop behaviours*, where continuous user input drives the behaviour" [21, pp. 31] without coding. This goes beyond prior prototyping tools (e.g. Exemplar [22]) that focus on supporting designers in authoring interactive behaviours based on discrete events extracted from continuous sensor data [21, pp. 99]. Moreover, Astral's continuous behaviours focus on *how* the input becomes animated as output *while the action takes place*.

## RELATED WORK AND GOALS

The goal of our prototyping tool is to author nuanced interactive behaviour on mobile devices. In this paper, we consider 'interactive behaviour' as the 'feel' of a prototype [49] that cannot be easily conveyed through a physical sketch and tends to require programming. 'Feel' emerges from not only the outputs once interactions happen, but also *as* interactions happen – a dynamic interplay discussed in past work on progressive feedback [62] and animation applications [18, 19].

We situate our research among extensive prior work on toolkits and prototyping tools that help interaction designers define interactive behavior. The specific approach of our prototyping tool leverages and extends past work on streaming sensor inputs, mirroring display outputs, and remapping inputs across devices. Based on previous work, we identify a series of Design Goals (***DG***) for Astral.

### Prototyping Tools

We draw heavily on prior work in prototyping tools where designers can quickly customize interactive behaviours (e.g. Adobe Flash [65]) and work with sensor input (e.g. Exemplar [22]), while leveraging existing applications (e.g. MaKey MaKey [4]), thus reducing the need to program. Astral

shares many goals of *Programming by Demonstration* approaches, including Exemplar [22], Improv [8], Gesture Morpher [54], Topaz [45], and SugiLite [36]. Unlike these approaches, however, Astral's emphasis is not on recognition of actions and generalization from examples, but on providing designers with a means for open-ended authoring of continuous behaviours.

*Mobile Prototyping*

Fast prototyping not only relies on expressiveness, but also how quickly designers can preview and evaluate designs. Many interface prototyping tools feature live prototyping to help designers create interactive applications in both mobile contexts [1, 42, 44, 53] and physical computing contexts [22, 23, 32]. Gummy Live [42] allows designers to create mobile interfaces live and see them reflected in the mobile device ready for modification. Similarly, de Sá et al. [53] created a tool that transitions from sketches on the target mobile device to Wizard of Oz [29] and higher fidelity prototypes. Thus, our first goal is to create a tool that ***prototypes live interactive behavior on the target device (DG1).*** We further support liveness by running rules in real-time, including live previews as they are created.

*Physical Prototyping*

Mobile devices and smart objects often contain multiple sensors, and designers often prototype behaviours based on sensor data. In physical computing, authoring is channeled through specialized hardware (e.g. [4, 15, 61, 67]). Arduino [66] in particular allows embedding multiple sensors into objects. The cost is that circuits and code exploiting the raw sensor data must be built from scratch, which can be difficult for novices, and introduces more opportunities for bugs [5]. Alternatively, designers can repurpose existing sensors and simplify authoring via prescribed interpretations for the sensor data [32, 56]. Exemplar [22] shows how visualizations can help designers to make sense of sensor data. Thus, our second goal is to ***provide an end-user interface that allows designers to explore variations among mobile sensors (DG2).*** We achieve this goal and extend prior work by our use of visualizations. First, we use live visualizations tailored to each sensor type. The sensor data can be manipulated to select its values and parameters for mapping. Second, a live visualization lists all sensor data, which is recorded alongside a synchronized video capture. As described later, these methods help increase expressive match [50].

### Extending Existing Infrastructures

When working on top of existing infrastructures, toolkits can leverage existing functionality to quickly explore new types of interactions. Olsen [50] discusses how working with common infrastructures enables new technology combinations to support new solutions. For example, when pen input behaves as mouse input, mouse-based applications can now be used with a pen as the input device. Many prototyping tools in the research literature use this approach. Exemplar [22] and MaKey MaKey [4] support remapping sensor input into mouse and keyboard events. ICon is a toolkit and editor [10] to create input-reconfigurable interactive applications.

More general-purpose tools look to augment existing desktop applications. For instance, Transmogrifiers [6] shows how dynamic desktop content (e.g. images, websites, videos) can be transformed on-the-fly to create free-form visualizations. Prefab [9] treats the desktop as a set of pixels, which can be reverse-engineered to enable new behaviour implementations on top of already existing interfaces (e.g. Bubble Cursor [17] on default menus). We extend the idea of working with the existing ecosystem of a desktop with its installed software and the native infrastructure (i.e. keyboard and mouse events). Instead of augmenting interaction with the existing application (as in Prefab [9]), we repurpose applications to author the mobile interactions.

Some tools extend interaction across multiple devices (commercial examples include TeamViewer [69] and VNC [52]). Semantic Snarfing [47] showed how a mobile device could act as a laser pointer to retrieve contents of a desktop. Myers [46] also examined how mobile devices could act as additional inputs to PCs (e.g. as a number pad). These systems stream mobile inputs (e.g. PDA pen events) to interact with the target application. Screen mirroring has a long history in HCI, dating back to systems such as Chameleon [12]. Gummy-Live [42] and D-Macs [43] leverage screen mirroring so designers can bridge the desktop's design environment and working with the target device. Montage [35] superimposes digital animated sketches on top of video prototypes. Virtual Projection [3] mirrors and augments a desktop's content and provides additional interaction opportunities. WinCuts [59] uses real-time image-based mirroring which extends to multiple devices without support for input redirection from the remote machines.

Building on these ideas of repurposing keyboard and mouse events, streaming mobile input, and mirroring screen contents, our third goal is to ***support the use of existing, familiar applications for prototyping (DG3)***. Astral enables designers to use their own workflows and remap sensor input (e.g. accelerometer, touch events) into keyboard and mouse input that can control *any* familiar desktop application to prototype mobile interactions.

### Continuous Animation

For designers to prototype interactive behaviors, they must not only be able to see continuous live effects from their input (***DG1***), but also examine and modify *how* those effects take place atop familiar applications ***(DG3)***. Thus, Astral needs to support ***continuous animation in direct response to sensor changes as they occur (DG4)*** if it is to truly prototype nuanced interactive behaviors. One way to achieve this is through *easings*. Easing is a term used by Adobe Flash [65] to refer to the slow-in and slow-out principle of animation [60], in which the number of in-between frames are increased or decreased at keyframes between poses to create the illusion that an object is speeding up or slowing down. Adobe Flash incorporated easings as a default linear tween that could be applied to motion tweens. Penner [51] created scripts for Flash to change the character of the easing through

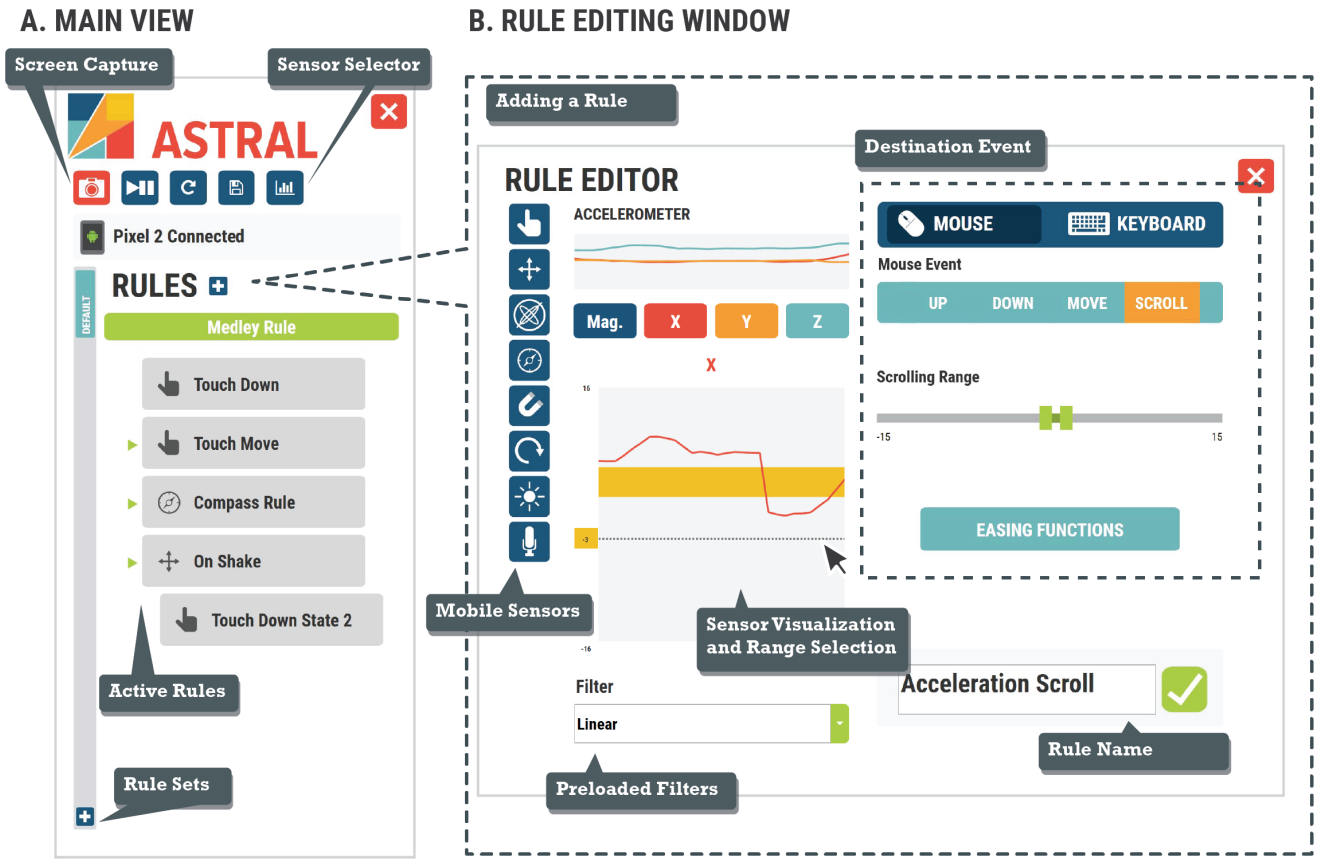


**Figure 2. Annotated Astral Interface. (a) Main window streams content to mobile device and displays active rules; (b) When adding rules, the interface shows an interactive visualization from which designers select the range of sensor values to for input remapping.**

mathematical functions. We leverage prior work that animates as a function of continuous sensor input (e.g. [2, 11, 38]) and extend Penner's easing functions [51] – instead of it being a function of time, they work as a function of continuous input (e.g. as done in [7, 18, 33, 40]) and we support their authoring as a way to fine-tune animations *as* continuous sensor-based interactions happen. The easing functions can produce aesthetic experiences, as well as more utilitarian elements (e.g. balancing the sensitivity of an input's effect). These continuous animations with easing functions are not explored in prior programming by demonstration approaches, which tend to favour recognition of discrete events from continuous sensor inputs (e.g. [22]). Astral extends this by authoring of interactive behaviour where continuous sensor input drives continuous output without writing code.

To recapitulate, Astral extends previous approaches by combining existing techniques of mirroring, streaming and remapping to feed into new building blocks: the creation of small, self-contained rules that drive a lively and animated prototype. These rules allow repurposing of familiar desktop applications in ways that have not been seen before.

## WORKING WITH ASTRAL

The overarching idea behind Astral (as shown in Figure 1) is to allow designers to quickly prototype interactive behaviours on mobile devices. To ensure that designers can use or repurpose familiar desktop applications to author and test mobile interactive behaviours (**DG3**), we designed Astral as a desktop server that communicates with a client running on designer's target mobile device.

### Main Interface

Once the mobile client connects to the desktop application, the main view provides access to all of the functionality. We next describe the different functions provided in the main interface (Figure 2a), which are later addressed in more detail.

***Mirror Desktop Contents.*** Designers can choose the region of the desktop that should be mirrored to the mobile client.

***Specify Input Remapping through Rules.*** Designers can author the intended interactive behaviour through the use of rules and rulesets (see following sections).

***Combine Rules into Rulesets.*** Active (authored) rules are shown on the screen, which can be edited via double clicking. Rules are by default added to the currently active ruleset.

***Visualize Sensor Data in the Sensor Selector.*** Designers can view all mobile sensors concurrently together with a video feed, record, playback and convert to rules.

### Mirroring Desktop Contents

Clicking on the camera icon (Figure 2a), designers can mirror display contents onto the connected mobile device. An

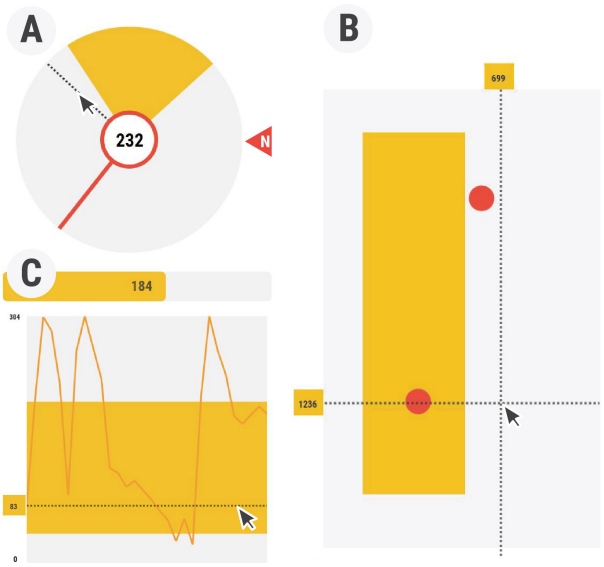


**Figure 3. Astral provides interactive visualizations for different sensors: (a) compass, (b) touchscreen, (c) ambient light. Selected areas of interest are highlighted in yellow.**

overlay region is shown, which the designer can move and scale to mirror onto the mobile device. We expect this region to typically contain the intended visual output created through the designer's preferred application. The selection window contents are mirrored to the mobile client live.

### Specifying Input Remapping through Rules

Once content is streamed to the mobile device, designers can author an interactive behaviour by defining a *rule*. A rule is a software abstraction that contains information as to how mobile sensor data is mapped to keyboard and mouse events. Each rule holds a sensor type, a range of values to which the mobile data is compared, and a mapping. Mappings encapsulate a *source* (mobile sensor input) and a *destination* (desktop mouse or keyboard event). To create a rule, the designer clicks on the 'plus' sign to open the *Rule Editor* – a guided interface to author or edit an input remapping rule. A particular configuration example is shown in Figure 2b.

***Selecting a Source: Sensor and Range of Values.*** The Rule Editor shows a list of sensors provided by the mobile device. A designer can choose the individual *sensor of interest* (Figure 2b side panel) to define the rule. Clicking on a sensor icon reveals a live visualization of the sensor and its values to help the designer understand (1) the particular sensor's response as the device is being manipulated (Figure 3), and (2) whether the sensor is appropriate to use. The visualization is tailored to the selected sensor (and its parameters/individual data) to provide higher expressive match [50]. In the case of the accelerometer (Figure 2b), the designer can select which parameter to observe (e.g. the *x*-dimension). They can then constrain the sensor to a range of values (e.g. between 5 $m/s^2$ and 5 $m/s^2$). Sensor readings can be further transformed by applying prepackaged filters (e.g. extracting gravity and linear acceleration values from the acceleration).

***Remapping the Source to a Destination Desktop Input.*** The designer can now map the mobile device sensor and its range of values to a *desktop (mouse or keyboard) input*, also consisting of a type of input and a range of values. Astral then interpolates between these two ranges of values. For example, when mapping to a mouse-move event, the designer can specify the destination range of pixel-coordinates by manipulating a rectangular selection. For mouse wheel events, inputs map to a variable mouse wheel scrolling range, as shown in Figure 2-B (Windows default scrolling: 120 pixels per step). For keyboard events, designers can specify an event (i.e. key down, key press, or key up) and the associated key (e.g. arrow left, spacebar). Keys can be typed or selected from a list of operating system defined keys (e.g. volume controls, media playback). Through keyboard remapping, the system can also trigger hotkeys to the active application.

We support *discrete* and *continuous* inputs (as categorized by Exemplar [22]) through consistent abstractions. Designers always select a range of values from the *source* input (Figure 3), and map it to a range of values of a *destination* mouse or keyboard event. The system automatically maps the values from one range to the other. If a continuous sensor (e.g. a range of values of the compass) is mapped to a continuous destination (e.g. a mouse-move with range of coordinates), the values are interpolated (which can be further altered via inversion or easing functions). In the case where either *source* or *destination* inputs only provide two values, the system still performs the interpolation. For example, a proximity sensor (with values 0 or 1) mapped to a mouse-move will only (abruptly) move to the beginning or end of the range of mouse coordinates. In such a case, perhaps a mapping to another input, such as a key-press might be more sensible, as the mapping will be one-to-one, if the sensor reads 1, the command for the key is pressed, otherwise the key is released. On the other hand, if the sensor values are continuous (e.g. range within accelerometer-x) and mapped to a binary input (e.g. key-press), the key down is triggered


**Figure 4. Astral's Sensor Selector records sensor along with a webcam video feed. A range of values can be selected (green) to open the Rule Editor with the sensor and values predefined.**

once the sensor value enters a selection range that is specified by the designer (orange range in Figure 3-C), and releasing via key up will be triggered once the sensor value exits that selection. Note that Astral treats key-press and mouse-click as special events in that they combine operations for pressing (down) and releasing (up) to facilitate these common operations. Mouse/key down and up events only trigger the single event when the source sensor value enters the selected range.

***Easing Functions.*** When mapping continuous device sensor input to desktop inputs, Astral allows designers to apply easing functions [51]. A rule defines a range as a source selection (e.g. accelerometer's low and high values) and a destination selection (e.g. mouse coordinates).

The authoring process is dynamic: designers can immediately view, test and modify rules as they author or edit them. If they want to stop the rule from running (e.g. because the mobile device input is taking over the mouse cursor), they can press the *'escape'* key to pause or play the live mapping. When the designer is finished, they can name the rule and add it to the active ruleset in the main window.

### Merging Several Rules into Rulesets

A behaviour may often require several rules, potentially using different sensors. Astral adds an additional layer of abstraction, *rulesets*, to support combining rules. If a ruleset is active, rules within that set will execute as long as the mobile device streams sensor data. This can be paused with the play/pause button, or by pressing the 'escape' key.

To test variations of interactive behaviours, designers can create multiple rulesets and switch between them at any time. When there is an active ruleset, a newly created rule will be added to that set and stacked vertically.

#### *Deciding When Rules are Triggered*

Rules are a minimal unit of mapping a *source* to a *destination* input, thus supporting further re-combinations. Additional structures can expand ruleset expressiveness.

***Conditional (When).*** When a device input either meets a condition (e.g. values within a selected range), rules inside a conditional structure are activated. Conditional structures are always listening for input, and as long as the condition (or its negation) is met, all contained rules will execute. Thus, one can implement techniques such as the clutch mechanism in tilt-to-zoom [24] by nesting two conditionals (i.e. once the conditions of *touch is down* and *not touch move* are met, it is possible to interactively map *accelerometer Y* data from the device to *mouse scroll up/down* desktop input).

***Sequence (Next).*** A sequence defines a chain of rule transitions (e.g. moving between different interface screens, forcing order between rules). After a rule in a sequence is executed, it becomes inactive and the following rule becomes active. Each rule in a sequence can mirror different portions of the desktop screen. Through sequence structures, Astral can approximate state-based approaches (as done by e.g. InVision) without explicitly implementing states.

***Medley.*** A medley switches the currently active ruleset to the next when a device input meets a condition. Designers can define a single medley at a time. The idea behind medley rules is to quickly switch to and thus test different variations of a prototype as part of *getting the right design* [16].

### Sensor Selector

We previously mentioned that Astral allows designers to disambiguate between multiple sensors. The Sensor Selector provides an overview of values from all available sensors as stacked line charts (Figure 4). By pressing the record button, the system records a webcam view that is synchronized with the different sensor data. Designers can go through the feed and see both video of the performed action and a visualization of the corresponding sensor data. Designers can then scrub with the mouse to select the area of interest at which the desired action takes place. From that selection, designers can see all sensors that reacted, and select a specific sensor to create a rule. The system will open the Rule Editor with the sensor and its recorded ranges already selected. Having all sensors displayed together with the webcam view can help designers select the relevant sensor to use.

## USAGE SCENARIO: CREATING A LEVEL

To demonstrate how a designer might work with Astral to author an interactive behaviour, we describe a simple scenario. A designer aims to create a level (akin to a carpenter's level) on a phone (Figure 5). In the interface, a bubble is centered on the screen when the phone is level and moves to corresponding sides when not level. Prototyping this type of nuanced interactive behaviour at this fidelity would ordinarily require extensive programming. In this scenario, we showcase how Adobe Illustrator and AfterEffects – familiar image and video applications to a designer [39] – can realize nuanced behaviours. Below, and in our video figure, we illustrate our scenario and note the duration of each step.

### Step 0: Illustrator and AfterEffects

The designer first uses Adobe Illustrator to create a level illustration, where the "bubble" is extracted as a separate layer that can be masked and animated (15 minutes). The designer

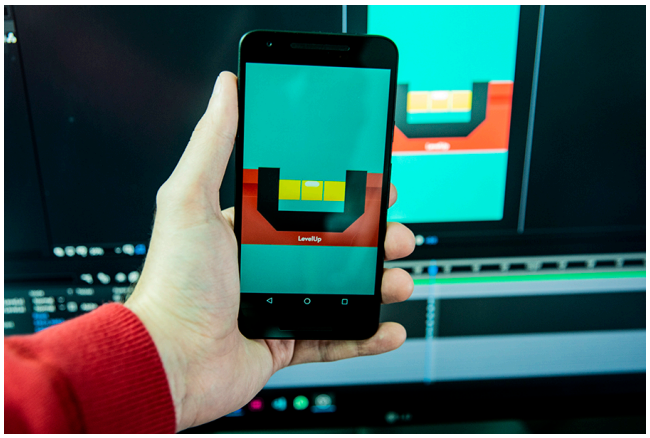

**Figure 5. Level prototype. Designers can create a custom mapping of a phone's acceleration values to a mouse move event which scrubs through an Adobe AfterEffects Timeline.**

then imports the Illustrator file into Adobe AfterEffects and creates a simple linear animation in which the bubble moves from one end of the level to the other as the video progresses through its timeline (7 minutes). At this point, the designer has a video prototype that can describe *what* happens, but not *how* it happens. Astral is needed to transform this desktop video into an interactive prototype on the mobile device.

#### Step 1: Starting Astral

The designer launches Astral on the desktop (Figure 2a) and connects the mobile device to it. The designer clicks the camera icon to select a region of the desktop to mirror onto the device. The designer selects the output video in the AfterEffects window, which appears live on the device (1 minute).

#### Step 2: Sensor Selector

The designer wants the interaction to play out when tilting the phone from side to side in a portrait orientation. Unsure of which sensor might be used for this, the designer opens the Sensor Selector (Figure 4) and records all available sensor values. The designer holds the device in view of the webcam and tilts the device side to side (2 minutes). The designer then plays back the video and sensor recording, to narrow down what happens as the device motion takes place. They find that *Linear Acceleration X* and *Linear Acceleration Y* both react to tilts, but that *Linear Acceleration Y* also triggers when tilting the device forward and back. The designer right clicks on *Linear Acceleration X* and clicks on the "Create Rule" option, which opens the rule editor (1 minute).

#### Step 3: Rule Editor

The Rule Editor (Figure 2b) automatically selects *Linear Accelerometer X* as its active sensor parameter, and already has a defined range based on the readings from the Sensor Selector. The designer repeats the desired behavior (side to side motion) to adjust the acceleration range (1 minute).

#### Step 4: Mapping Mouse Coordinates to the AfterEffects Timeline

The designer now uses input remapping to specify *how* the interaction takes place: moving the device from side to side is remapped to mouse actions that scrub through the video timeline so that the level's bubble reacts accordingly. The designer creates a mapping by clicking on '*Mouse'* and selecting the *move* event. The designer next defines a mapping area which they assign to a rectangle overlaying the AfterEffects Timeline and ticks the checkbox so that the mouse performs a mouse *down* (holding) whenever the *move* event takes place. Because of the immediate preview, moving the phone already causes the mouse to move (which can be activated or deactivated from anywhere in the operating system using the *escape* key). As the prototype is already interactive via its live preview, the designer immediately sees the effects (both input and output) in the mobile device (2 minutes).

#### Step 5: Fine-Tuning through Easing Functions

When the interaction is tested, the designer might find that it does not respond as desired, as it is very easy for the level to go quickly from one side to another. One way to mitigate this is through an inverse cubic-in-out easing – which would slow down the animation towards the middle of the timeline, and speed up the animation towards the edges of the timeline, making the bubble remain level for longer. The designer can try different easing functions provided by Astral to balance a correct indication of when the level suggests it is level with a reaction that feels engaging (it only takes a few seconds to apply an easing function). Through easing, the designer is able to fine-tune the animation qualities of the interaction. This can take as long as the designer wishes to fine-tune the interaction. The designer may also decide to readjust the input parameters or the mouse region for further fine-tuning.

## PROTOTYPES AND INTERACTION SCENARIOS

We implemented a series of novel and replicated prototypes using Astral, which help convey Astral's threshold, ceiling, and expressiveness [48] and show how Astral might support different interaction design tasks. These scenarios show some of the ways in which applications might be repurposed.

#### Video-Based Prototyping

Both our own experiences and past literature have shown designers' inclination towards working with high-fidelity video to convey prototype ideas to developers [39, 63, 64]. While video can show state-based animations, it does not show how interaction affects the timing of these animations as continuous inputs are taking place. With Astral, designers can map sensors to mouse events that scrub through portions of the timeline in their preferred video editor. With this approach, skilled designers can achieve rich visuals with detailed effects (e.g. changing size and shape of the level's bubble as it moves) which would otherwise be quite complex to program.

***Level Mobile Phone App.*** The level was described in the usage scenario section. With the level prototype, we emphasize (1) how the Sensor Selector can help designers determine which sensor corresponds to an action (in this case determining tilt by acceleration); and (2) the power of easing functions to change the 'feel' [49] of an interactive behaviour.

***Compass.*** We created a simple animation of a compass needle rotating 360 degrees, including a separately-animated needle shadow that creates a three-dimensional effect when in motion. We mapped the angle of the device's compass sensor to the position on the video timeline.

***Quick Settings.*** The Android Quick Settings menu contains a nuanced animation where multiple icons change size, position, and opacity, to reveal available operating system functions to a mobile user. With Astral, we are able to map a downward sliding gesture to progressively reveal controls. Furthermore, we can add an additional interaction of controlling the screen brightness by mapping a side swipe on the top of the screen to another portion of the timeline in which the screen fades to black. This shows how even within video timelines Astral can support multiple interactions.

#### Authoring Open-Ended Interaction Techniques

With Astral, it is also possible to prototype interaction techniques that provide more open-ended ways of interaction than the video-based prototypes.

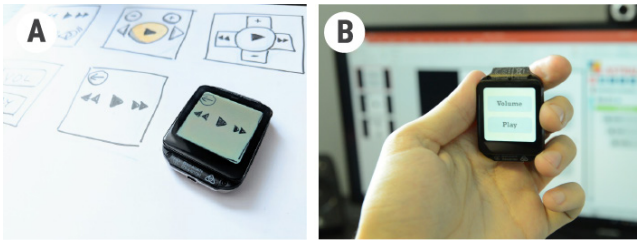


**Figure 6. Astral supports the design process in all stages by allowing (a) on-device rapid creation of interactive sketches, (b) using slideshows to transition between states.**

***Tilt to Move.*** We used Astral to create a one-handed map navigation by mapping the different tilt directions from a phone's accelerometer data to the cardinal arrow keys in Google Maps. The rules are set so that key commands are triggered when the acceleration crosses a certain range (x: 4 to 7 triggers right, x: -4 to -7 triggers left, y: 4 to 7 triggers down, y: -4 to -7 triggers up). Because Astral is using a keypress event, the mapping initiates a key down when the accelerometer enters the specified range, and a key up when leaving the range. This scenario replicates an example from d.tools [23] that originally required programming to realize the tilt-based map navigation. In contrast, the Astral version leverages input remapping and avoids the need to write code.

***Tilt to Zoom.*** We implemented tilt-to-zoom [24], where a designer can both pan through a map using touch, and zoom in and out via tilting provided that there is also a touch down event (their finger acts as a clutch). This is achieved using conditional constructs. A *touch down* conditional becomes active if touch is down on the device. It contains another nested condition that checks whether *touch move* is *not* taking place. The rule within this nested conditional maps the accelerometer's *y-dimension* to mouse scrolling (up or down). This prototype replicates an example from Hinckley et al. [24], incorporating the concept of *motion in touch* – mapping more than one sensor to a single function.

### Prototyping Multiple Alternatives

Astral supports the exploration of different design solutions (*getting the right design* [16]). This motivated our *medley rule* which switches between active rulesets. Designers can sequentially test a set of prototype alternatives. Astral can thus support experimentation with any variation within rules, including sensors, thresholds, easings, or desktop inputs.

***Input Variations in a Mobile Game.*** The mobile platformer game Flappy Bird features a bird that flaps its wings when tapping the screen with the goal of making the bird fly through pipes. A web version (http://flappybird.io) allows players to use the spacebar, a mouse, or touch if using a touch-enabled device. Running the game on a desktop computer, we mapped different mobile sensors to a spacebar keypress (illustrated in Figure 1) so that the bird flaps when tapping, when blowing onto the microphone, and when shaking. By creating a medley rule, we can quickly switch between active rulesets to explore different forms of interaction – in this case whenever the light sensor is covered.

### Iterative Prototyping at Multiple Fidelities

Since Astral remaps inputs and supports mobile sensors to map to any key, we can work with multiple applications at different stages of the design process and support different tasks and specialized tools – wireframing and walkthroughs, transitions between states / flow (similar to d.tools [23]), or working with more sophisticated programming platforms that may not be available for mobile prototyping. To realize these examples, Astral mainly relies on *sequences.*

***Music Controller Sketches.*** Using a default image viewer, we can scan or photograph an interface sketch and immediately view it on the mobile device (Figure 6a). Designers can emulate states by chaining multiple rules with the sequence construct. Each rule moves the streamed region to different parts of the image (i.e. the screen drawings) depending on the tap interactions that may take place. By previewing the sketches on the target device – here a watch – designers can make early decisions such as defining correct button sizes.

***Music Controller PowerPoint Mock-up.*** Presentation software such as PowerPoint and Keynote remain relevant for mocking up interfaces and wireframes [39, 63, 64]. With Astral, it is possible to use mock-ups created with these applications to show what seems like a button press on the watch (given the streamed visual) and move to another part of the slideshow by perform a click event on different parts of the slide thumbnail preview (Figure 6b). Thus, one can easily test the flow between different interface screens.

***HTML Prototype.*** Some designers are comfortable programming HTML [63, 64]. They can map touch events one-to-one on a desktop browser window and test it on a smartwatch.

### Authoring Smart Object Behaviours

Designers may also leverage Astral to explore behaviours on smart objects and appliances, as well as some degree of IoT. Using Soul-Body prototyping [32], designers can repurpose phones and watches in new and interesting ways.

***Smart Speaker Animations.*** Using video editing applications, one can author nuanced animated responses that a smart home speaker might perform. We created a smart speaker prototype by placing a smart watch inside a mug with a 3D printed tray and light diffuser (Figure 7a). Astral supports speech recognition through the built-in Microsoft speech API, so one could also explore different kinds of animations depending on different voice commands.

***3D Printed 'Smart' Level.*** We recreated the 3D-printed level from Pineal [32] by reusing the level mobile phone app prototype from earlier in a smartwatch enclosed in a larger 3D print (Figure 7b). This shows how Astral can also adapt prototypes to different devices and form factors.

## IMPLEMENTATION

Astral is designed to work with one mobile device per desktop Astral client, which constrains and simplifies the workflow. This is tied to a technical limitation of desktops, as mouse and keyboard commands only can be sent to a single

focused program. The desktop client of Astral is implemented using C# and WPF, while the mobile applications are written in C# Xamarin to allow cross-platform mobile development (iOS, Android, Android Wear). To reuse code and quickly adapt to newly added sensors of future devices, we developed all communication aspects in shared code, which uses the .NET Standard 2.0 (see below for details).

***Device Modules.*** We created classes in shared code for each of the mobile device's features (e.g. accelerometer, microphone, or display), which we call *device modules*. The mobile device instantiates all modules it is equipped with when the application starts. Once the device connects to the Astral desktop, it sends a list of all available modules to the desktop. The desktop then creates the same modules to access the sensors by proxy, as if they were local sensors. Each module updates its values with newly measured sensor data. Modules trigger an event in code once values have been updated.

***Data Exchange between Devices.*** Because the desktop and the client are not running on the same machine, device modules handle the internal network communication. For sensor data coming from the mobile device, this works as follows: (1) the mobile device records the respective sensor data natively (i.e. iOS or Android specific); (2) it then updates the module using a device-independent abstraction of the measured data (e.g. three floating-point numbers for the accelerometer); (3) the module sends this data as bytes (using a unique identifier) over the network; (4) the module on the desktop unpacks the message and triggers an event; (5) if the Astral desktop client subscribes to the event, it receives the sensor data, and sends the update to rules using that sensor.

Mirroring desktop contents works similarly, except that the desktop client updates the display device module. To speed up the transmission of images, we detect changes through image differencing, compress the areas that changed (JPEG), and only transmit these image patches.

***Performance.*** We use wireless LAN via TCP for connectivity between devices. We tested Astral on multiple phones (Nexus 5 and 5X, iPhones 6, 7 and 8, Pixel 2) and one smartwatch (Sony Smartwatch 3). Image transmission is at interactive framerates – 50 fps on iOS, 25 fps on Android. This is concurrent with mobile sensor data streaming to the desktop, yet only if the desktop actually requires a specific sensor (i.e., a Rule or the Sensor Viewer is using that sensor). We stream sensor data in real-time but we restrict the rate to 100 fps to ensure high transmission rates in both directions. During testing and creation of our prototypes, we did not experience significant delays transferring data from multiple sensors.

## DISCUSSION

In the process of creating Astral, we found much room for critical reflection, both in terms of the extent to which we achieved our goals, and limitations we tried to address.

### Astral and Designers

In creating Astral, we wanted to provide designers with more ways to express interactive behaviours – supporting the use of sensors and physical actions, and providing a means to create smooth, reactive outputs. In that sense, these are new activities that designers may not have the means to, or not frequently need to perform today in their everyday jobs, but we are seeing systems increasingly using these types of behaviours (e.g. Android Quick Settings, Slide to Unlock). We also see these fluid behaviours in many emerging interactive devices (e.g. smart speakers), with a lack of tools to facilitate prototyping these behaviours. Such fluid behaviours are important, as they help people understand what their products are doing, might do, or have done [62], and communicate that the product has been designed with care [55].

### Target Audience

As we designed Astral, we kept in mind that our target audience will likely not have a strong technical background, as suggested by prior work [39, 49, 63, 64] and our experience. Our prototypes reflect a variety of interaction design tasks, some featuring very novel applications. We also carefully thought about *expressive match* [50]: tailoring visualizations to particular mobile sensor data, creating rules and simplifying mappings to keyboard and mouse events. We free designers from thinking about the nature of inputs by removing the distinction between discrete and continuous values.

### State Models

Astral is not intended to replace existing prototyping software, but to instead provide an alternative approach. State models are a common prototyping strategy, as they can quite intuitively describe the flow of the interaction. In providing paths of least resistance [48], we approximated this state-based approach through the *sequence* structure and by leveraging features in applications that can emulate states (e.g. mapping behaviour to different slides in PowerPoint or to portions of a video timeline). While state models support more complex state-based applications, they lack support for authoring detailed dynamic aspects of user interface behaviour, instead favouring a trigger-action interaction model. An interesting idea for future work would be to combine state models with our input remapping and easing functions in Astral. This would enable more complex states while still facilitating prototyping the 'feel' aspects of interactive behaviour.


**Figure 7. Our Soul–Body Prototypes working with Astral: (a) Smart Speaker, prototyped by 3D printing a watch tray and light diffuser fitted inside a travel mug; (b) 3D Printed 'Smart' Level that reuses our phone app on an enclosed smartwatch.**

### Expressive Leverage and Flexibility

Astral allows designers to quickly get started (*threshold*) and achieve fairly expressive results (*ceiling*). However, we only examined a small subset of the range of interaction possibilities with these types of inputs and this type of tool. When leveraging mouse events, the desktop tool needs interactive regions that are visible in the windows' viewports to be a viable target for input remapping. For example, one might navigate between PowerPoint slides on the thumbnail view, or by playing the slideshow, but it is not possible to go to a specific slide without a macro.

In spite of the limitations of mouse and keyboard events, using Astral on top of an existing tool empowers designers to repurposing the underlying desktop tool in new ways. In that sense, Astral augments existing tools with new *paths of least resistance* [48]. For example, the video prototype becomes interactive thanks to Adobe Premiere's timeline and live preview – Astral provides a means to define how the mobile sensors can manipulate the timeline and the preview to repurpose Premiere for testing continuous behaviours.

### Overcoming 'Input Locks'

In a few cases, we observed issues with input locks when using Astral. Once a mouse move event is selected, the mouse starts reacting to the incoming sensor data. It sometimes became impossible to move the cursor with the physical mouse. To preserve the ability to test behaviours live as they are authored, we remedied this by adding a toggle with the 'escape' key to enable or disable the live preview.

### Device Relativism and Saving Prototypes

Mappings of mouse and screen coordinates may not carry across different computers with different resolutions. One way to address this is to use device coordinates. Another potential concern is that window sizes are not fixed, so once the workspace has changed the mappings may no longer work. There are workarounds to the latter concern: it is possible to store the position and sizes of the windows and associate them to the rules, so that when a ruleset executes it adjusts the window sizes. Mobile phones and smartwatches also have a wide variation within their resolutions and sensors. Additionally, some sensors may not be available on each device, and some sensors may have device-specific readings. One possible extension to allow designers to resume their work is for Astral is to record and store the rules, mirrored screen and sensor values. This can further scale to work across different mobile devices (resolutions and sensor ranges) to support sharing and testing of prototypes.

### Evaluation Approach

There are various strategies for evaluating toolkits [34]. Of these, we use *evaluation by demonstration* as our primary method. Our prototypes represent both novel and replicated systems from past research [23, 24, 32, 46, 58, 59] – which reflect how Astral can achieve results that might be difficult to create otherwise, as well as ensuring prior *paths of least resistance* [48] can still be accommodated. Our usage scenario provides a perspective on how designers might work with Astral while conveying some of its *threshold* and *ceiling* [48]. We benchmarked the performance of the image transfer, which reached up to 25 fps on Android, and 50 fps on iOS. Finally, in this discussion, we took a reflective stance guided by prior lessons by Olsen [50] and Myers et al. [48].

We deliberately did not pursue a lab usability study. A usability evaluation would be inappropriate given that Astral is not a walk-up and use system and the paths of interaction are very open-ended [50]. Furthermore, a lab study would sacrifice realism [41]. First, designers each have different applications and computer setups which cannot be reflected in a lab setting. Second, Astral provides an alternative way to think about prototyping, which requires time to internalize. Finally, short tasks can lead to the usability trap [50], or test tasks we know Astral can succeed at, thus leading to unfair comparisons or weak generalizations from the current implementation rather than the concept as a whole [14]. Open-ended tasks would require designers to envision ideas ahead of time (thus requiring an understanding of what Astral can do) – it would be unreasonable to request a design on the spot. An observational field study is beyond the scope of this paper, but would help us understand how Astral's workflow fits interaction designers and affects what they can create, and how designers evolve their use of familiar applications over time to exploit Astral's capabilities.

## ACKNOWLEDGEMENTS

This research was supported by NSERC, NSERC PGS-D, AITF, Smart Technologies, the Adobe Research Fellowship, the Isaac Walton Killam Scholarship and the Office of the Vice President (Research) at the University of Calgary.

## CONCLUSION

We presented Astral, a prototyping tool that addresses challenges in interactive behaviour design. By (1) mirroring contents from a desktop display, (2) streaming sensor data from a mobile device, and (3) allowing designers to map mobile sensors to mouse and keyboard events, a designer can choose a familiar desktop application to author mobile and smart object interactions. Through Astral, we provide visualizations to help designers make sense of mobile data where multiple sensors are feeding data at the same time. From it, they can explore, author and fine-tune dynamic behaviours (both as input happens, as well as once inputs happen). Finally, by leveraging mouse and keyboard events, designers can use or repurpose any familiar desktop application and have it readily available for live prototyping, thus multiplying the number of tools at designers' disposal. We demonstrated Astral's potential and workflow through a series of prototypes which encompass common interaction design activities as informed by prior explorations [39, 49, 63, 64] and our experience. We hope our initial exploration can propel designers' conversations around interactive behaviour and lower the challenges of transitioning from design to implementation.

## REFERENCES


[1] Daniel Ashbrook and Thad Starner. 2010. MAGIC: a motion gesture design tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10). ACM, New York, NY, USA, 2159-2168. https://doi.org/10.1145/1753326.1753653

[2] Olivier Bau and Wendy E. Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of ACM Symposium on User Interface Software and Technology* (UIST '08). ACM, New York, NY, USA, 37-46. https://doi.org/10.1145/1449715.1449724

[3] Dominikus Baur, Sebastian Boring, and Steven Feiner. 2012. Virtual projection: exploring optical projection as a metaphor for multi-device interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '12). ACM, New York, NY, USA, 1693-1702. http://dx.doi.org/10.1145/2207676.2208297

[4] Beginner's Mind Collective and David Shaw. 2012. Makey Makey: improvising tangible and nature-based user interfaces. In *Proceedings of the ACM International Conference on Tangible, Embedded and Embodied Interaction* (TEI '12). ACM, New York, NY, USA, 367-370. https://doi.org/10.1145/2148131.2148219

[5] Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '16). ACM, New York, NY, USA, 3485-3497. https://doi.org/10.1145/2858036.2858533

[6] John Brosz, Miguel A. Nacenta, Richard Pusch, Sheelagh Carpendale, and Christophe Hurter. 2013. Transmogrification: causal manipulation of visualizations. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '13). ACM, New York, NY, USA, 97-106. https://doi.org/10.1145/2501988.2502046

[7] Jesse Burstyn, Juan Pablo Carrascal, and Roel Vertegaal. 2016. Fitts' Law and the Effects of Input Mapping and Stiffness on Flexible Display Interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '16). ACM, New York, NY, USA, 3649-3658. https://doi.org/10.1145/2858036.2858383

[8] Xiang 'Anthony' Chen and Yang Li. 2017. Improv: An Input Framework for Improvising Cross-Device Interaction by Demonstration. *ACM Trans. Comput.-Hum. Interact.* 24, 2, Article 15 (April 2017), 21 pages. https://doi.org/10.1145/3057862

[9] Morgan Dixon, Alexander Nied, and James Fogarty. 2014. Prefab layers and prefab annotations: extensible pixel-based interpretation of graphical interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '14). ACM, New York, NY, USA, 221-230. https://doi.org/10.1145/2642918.2647412

[10] Pierre Dragicevic and Jean-Daniel Fekete. 2004. Support for input adaptability in the ICON toolkit. In *Proceedings of the International Conference on Multimodal Interfaces* (ICMI '04). ACM, New York, NY, USA, 212-219. http://dx.doi.org/10.1145/1027933.1027969

[11] Pierre Dragicevic, Gonzalo Ramos, Jacobo Bibliowitcz, Derek Nowrouzezahrai, Ravin Balakrishnan, and Karan Singh. 2008. Video browsing by direct manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '08). ACM, New York, NY, USA, 237-246. https://doi.org/10.1145/1357054.1357096

[12] George W. Fitzmaurice. 1993. Situated information spaces and spatially aware palmtop computers. *Commun. ACM* 36, 7 (July 1993), 39-49. http://dx.doi.org/10.1145/159544.159566

[13] Saul Greenberg. 2007. Toolkits and interface creativity. *Multimedia Tools and Applications*, *32*(2), Springer, 139-159. https://doi.org/10.1007/s11042-006-0062-y

[14] Saul Greenberg and Bill Buxton. 2008. Usability evaluation considered harmful (some of the time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '08). ACM, New York, NY, USA, 111-120. https://doi.org/10.1145/1357054.1357074

[15] Saul Greenberg and Chester Fitchett. 2001. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '01). ACM, New York, NY, USA, 209-218. http://dx.doi.org/10.1145/502348.502388

[16] Saul Greenberg, Sheelagh Carpendale, Nicolai Marquardt and Bill Buxton (2011). *Sketching User Experiences: The Workbook.* Elsevier.

[17] Tovi Grossman and Ravin Balakrishnan. 2005. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '05). ACM, New York, NY, USA, 281-290. http://dx.doi.org/10.1145/1054972.1055012

[18] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: sketch-

ing dynamic and interactive illustrations. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '14). ACM, New York, NY, USA, 395-405. https://doi.org/10.1145/2642918.2647375

[19] Rubaiat Habib Kazi, Tovi Grossman, Nobuyuki Umetani, and George Fitzmaurice. 2016. Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16). ACM, New York, NY, USA, 4599-4609. https://doi.org/10.1145/2858036.2858386

[20] Christopher Michael Hancock, 2003. *Real-time programming and the big ideas of computational literacy*. PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

[21] Björn Hartmann, 2009. *Gaining Design Insight through Interaction Prototyping Tools*. PhD Thesis, Stanford University.

[22] Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '07). ACM, New York, NY, USA, 145-154. https://doi.org/10.1145/1240624.1240646

[23] Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '06). ACM, New York, NY, USA, 299-308. https://doi.org/10.1145/1166253.1166300

[24] Ken Hinckley and Hyunyoung Song. 2011. Sensor synaesthesia: touch in motion, and motion in touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11). ACM, New York, NY, USA, 801-810. https://doi.org/10.1145/1978942.1979059

[25] Lars Erik Holmquist. 2005. Prototyping: generating ideas or cargo cult designs?. *interactions* 12, 2 (March 2005), 48-54. http://dx.doi.org/10.1145/1052438.1052465

[26] Kasper Hornbæk and Antti Oulasvirta. 2017. What Is Interaction?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '17). ACM, New York, NY, USA, 5040-5052. https://doi.org/10.1145/3025453.3025765

[27] Stephanie Houde, and Charles Hill. 1997. "What do prototypes prototype?" *Handbook of Human-Computer Interaction (Second Edition)*, 367-381.

[28] Steven Houben and Nicolai Marquardt. 2015. WatchConnect: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '15). ACM, New York, NY, USA, 1247-1256. https://doi.org/10.1145/2702123.2702215

[29] J. F. Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems*. 2, 1 (January 1984), 26-41. http://dx.doi.org/10.1145/357417.357420

[30] Ju-Whan Kim and Tek-Jin Nam. 2013. EventHurdle: supporting designers' exploratory interaction prototyping with gesture-based sensors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 267-276. https://doi.org/10.1145/2470654.2470691

[31] James Lin and James A. Landay. 2008. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '08). ACM, New York, NY, USA, 1313-1322. https://doi.org/10.1145/1357054.1357260

[32] David Ledo, Fraser Anderson, Ryan Schmidt, Lora Oehlberg, Saul Greenberg, and Tovi Grossman. 2017. Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '17). ACM, New York, NY, USA, 2583-2593. https://doi.org/10.1145/3025453.3025652

[33] David Ledo, Saul Greenberg, Nicolai Marquardt, and Sebastian Boring. 2015. Proxemic-Aware Controls: Designing Remote Controls for Ubiquitous Computing Ecologies. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services* (MobileHCI '15). ACM, New York, NY, USA, 187-198. https://doi.org/10.1145/2785830.2785871

[34] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '18). ACM, New York, NY, USA. https://doi.org/10.1145/3173574.3173610

[35] Germán Leiva and Michel Beaudouin-Lafon. 2018. Montage: A Video Prototyping System to

Reduce Re-Shooting and Increase Re-Usability. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)*. ACM, New York, NY, USA, 675-682. https://doi.org/10.1145/3242587.3242613

[36] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6038-6049. https://doi.org/10.1145/3025453.3025483

[37] John H. Maloney and Randall B. Smith. 1995. Directness and liveness in the morphic user interface construction environment. In *Proceedings of the ACM Symposium on User Interface and Software Technology* (UIST '95). ACM, New York, NY, USA, 21-28. http://dx.doi.org/10.1145/215585.215636

[38] Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. 2012. Gradual engagement: facilitating information exchange between digital devices as a function of proximity. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (ITS '12). ACM, New York, NY, USA, 31-40. https://doi.org/10.1145/2396636.2396642

[39] Nolwenn Maudet, Germán Leiva, Michel Beaudouin-Lafon, and Wendy Mackay. 2017. Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work and Social Computing* (CSCW '17). ACM, New York, NY, USA, 630-641. https://doi.org/10.1145/2998181.2998190

[40] Andrew McCaleb Reach, and Chris North. 2017. "The Signals and Systems Approach to Animation." arXiv preprint arXiv:1703.00521 (2017).

[41] Joseph McGrath. 1995. “Methodology Matters: Doing Research in the Behavioral and Social Sciences”. *Readings in Human–Computer Interaction*. 152-169.

[42] Jan Meskens, Kris Luyten, and Karin Coninx. 2009. Shortening user interface design iterations through realtime visualisation of design actions on the target device. In *IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC 2009), pp. 132-135. 10.1109/VLHCC.2009.5295281

[43] Jan Meskens, Kris Luyten, and Karin Coninx. 2010. D-Macs: building multi-device user interfaces by demonstrating, sharing and replaying design actions. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '10). ACM, New York, NY, USA, 129-138. https://doi.org/10.1145/1866029.1866051

[44] Jan Meskens, Jo Vermeulen, Kris Luyten, and Karin Coninx. 2008. Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (AVI '08). ACM, New York, NY, USA, 233-240. https://doi.org/10.1145/1385569.1385607

[45] Brad A. Myers. 1998. Scripting graphical applications by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 534-541. http://dx.doi.org/10.1145/274644.274716

[46] Brad Myers. 2002. Mobile devices for control. In *International Conference on Mobile Human-Computer Interaction*, Springer, Berlin, Heidelberg 1-8. https://doi.org/10.1007/3-540-45756-9_1

[47] Brad Myers, Choon Hong Peck, Jeffrey Nichols, Dave Kong, and Robert Miller. 2001. Interacting at a distance using semantic snarfing. In *International Conference on Ubiquitous Computing*. Springer, Berlin, Heidelberg pp. 305-314. https://doi.org/10.1007/3-540-45427-6_26

[48] Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction.* 7, 1 (March 2000), 3-28. http://dx.doi.org/10.1145/344949.344959

[49] Brad Myers, Sun Young Park, Yoko Nakano, Greg Mueller, and Andrew Ko. 2008. How designers design and program interactive behaviors. 2008. In Proc. *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pp. 177-184. 10.1109/VLHCC.2008.4639081

[50] Dan R. Olsen, Jr.. 2007. Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (UIST ‘07). ACM, New York, NY, USA, 251-258. https://doi.org/10.1145/1294211.1294256

[51] Robert Penner. *Robert Penner's Programming Macromedia Flash MX*. McGraw-Hill, Inc., 2002.

[52] Tristan Richardson, John Levine. 2011 The Remote Framebuffer Protocol (No. RFC 6143). Technical Report. https://tools.ietf.org/html/rfc6143

[53] Marco de Sá, Luís Carriço, Luís Duarte, and Tiago Reis. 2008. A mixed-fidelity prototyping tool for mobile devices. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (AVI

'08). ACM, New York, NY, USA, 225-232. https://doi.org/10.1145/1385569.1385606

[54] Ramik Sadana and Yang Li. 2016. Gesture morpher: video-based retargeting of multi-touch interactions. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*. ACM, New York, NY, USA, 227-232. https://doi.org/10.1145/2935334.2935391

[55] Dan Saffer. 2013. *Microinteractions: designing with details*. O'Reilly Media, Inc.

[56] Valkyrie Savage, Colin Chang, and Björn Hartmann. 2013. Sauron: embedded single-camera sensing of printed physical user interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '13). ACM, New York, NY, USA, 447-456. http://dx.doi.org/10.1145/2501988.2501992

[57] P. Frazer Seymour, Justin Matejka, Geoff Foulds, Ihor Petelycky, and Fraser Anderson. 2017. AMI: An Adaptable Music Interface to Support the Varying Needs of People with Dementia. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '17). ACM, New York, NY, USA, 150-154. https://doi.org/10.1145/3132525.3132557

[58] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips, and Nicolas Roussel. 2006. User interface façades: towards fully adaptable user interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (UIST '06). ACM, New York, NY, USA, 309-318. https://doi.org/10.1145/1166253.1166301

[59] Desney S. Tan, Brian Meyers, and Mary Czerwinski. 2004. WinCuts: manipulating arbitrary window regions for more effective use of screen space. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 1525-1528. https://doi.org/10.1145/985921.986106

[60] Frank Thomas, Ollie Johnston, and Disney Animation. 1981. *The Illusion of Life*. Abbeville Press, New York, NY, USA.

[61] Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. 2012 .NET Gadgeteer: A Platform for Custom Devices. In *Pervasive 2012.Lecture Notes in Computer Science*, vol 7319. Springer, Berlin, Heidelberg. 216-233 https://doi.org/10.1007/978-3-642-31205-2_14

[62] Jo Vermeulen, Kris Luyten, Karin Coninx, and Nicolai Marquardt. 2014. The design of slow-motion feedback. In *Proceedings of the SIGCHI Conference on Designing Interactive Systems* (DIS '14). ACM, New York, NY, USA, 267-270. https://doi.org/10.1145/2598510.2598604

[63] The Tools Designers are Using Today (2015 Survey) http://tools.subtraction.com/ – Accessed April 01, 2018.

[64] The 2017 Annual Design Tools Survey https://uxtools.co/survey-2017 – Accessed August 27, 2018.

[65] Adobe Animate (formerly Adobe Flash) https://www.adobe.com/ca/products/animate.html – Accessed September 20, 2018.

[66] Arduino http://arduino.cc – Accessed April 01, 2018.

[67] Microsoft MakeCode https://makecode.com/ – Accessed September 20, 2018.

[68] Nintendo Labo - https://labo.nintendo.com/ - Accessed September 20, 2018.

[69] TeamViewer http://www.teamviewer.com – accessed January 11, 2019