Apparition: Crowdsourced User Interfaces That Come To Life As You Sketch Them

Walter S. Lasecki
Computer Science Department
University of Rochester
wlasecki@cs.rochester.edu

Juho Kim
CSAIL
MIT
juhokim@mit.edu

Nicholas Rafter
Independent Researcher
nicholas.rafter@gmail.com

Onkur Sen
Computer Science Department
Stanford University
onkursen@gmail.com

Jeffrey P. Bigham
HCI and LT Institutes
Carnegie Mellon University
jbigham@cmu.edu

Michael S. Bernstein
Computer Science Department
Stanford University
msb@cs.stanford.edu

ABSTRACT

Prototyping allows designers to quickly iterate and gather feedback, but the time it takes to create even a Wizard-of-Oz prototype reduces the utility of the process. In this paper, we introduce crowdsourcing techniques and tools for prototyping interactive systems in the time it takes to describe the idea. Our Apparition system uses paid microtask crowds to make even hard-to-automate functions work immediately, allowing more fluid prototyping of interfaces that contain interactive elements and complex behaviors. As users sketch their interface and describe it aloud in natural language, crowd workers and sketch recognition algorithms translate the input into user interface elements, add animations, and provide Wizard-of-Oz functionality. We discuss how design teams can use our approach to reflect on prototypes or begin user studies within seconds, and how, over time, Apparition prototypes can become fully-implemented versions of the systems they simulate. Powering Apparition is the first self-coordinated, real-time crowdsourcing infrastructure. We anchor this infrastructure on a new, lightweight write-locking mechanism that workers can use to signal their intentions to each other.

Author Keywords

Rapid prototyping; crowdsourcing; human computation

ACM Classification Keywords

D.2.2 Design Tools and Techniques

INTRODUCTION

The effort required to produce an interactive prototype is a clear limiting factor in effective design practice. Prototyping grounds an iterative process of creation and reflection [28], where the prototype becomes a mechanism for continuous course correction and ideation. This process, which David Kelley terms "enlightened trial and error", is most effective when prototypes can be fluidly produced and evaluated. However, even the most rapid interactive prototype construction (e.g., Wizard-of-Oz prototyping) is easily an order of magnitude slower than envisioning through a sketch.

Sensing this disparity, foundational work nearly 20 years ago suggested that designers might be able to author interactive prototypes simply by sketching [14]. However, while these systems can transform simple elements such as rectangles into buttons, they stop short of capturing the full variety of interactive elements and behaviors a designer might envision. If a designer wants to create a simple video game character, make them follow a cursor, and destroy enemies by jumping on them, they are facing hours in a code editor.

In this paper, we introduce tools and techniques for prototyping interactive systems in the time it takes to simply describe the idea visually and verbally. Our system, called *Apparition*, uses paid crowds to make even hard-to-automate functions work immediately without the need for one-off control implementations, allowing fluid prototyping of interfaces containing interactive elements, complex animations, and intelligent feedback. As users sketch their interface and describe it aloud in natural language, crowd workers refine interface elements, add animations, and control the prototype's responses. Visual elements replace sketches within 8 seconds on average, and interactive behaviors function in 3 seconds. The speed of such prototypes makes it possible to create an interactive prototype moments after thinking of the idea.

For Apparition to work, a group of microtask workers must coordinate a complex set of goals and dependencies in real-time. *Apparition is the first crowdsourcing system to enable self-managing, real-time crowd coordination.* Unfortunately, direct approaches yield blocking behaviors where multiple workers all attempt the same task and conflict while other tasks remain untouched. Introducing managers slows the system beyond real-time responses. We introduce a lightweight write-locking mechanism that lets workers signal to others which elements they are editing and roles they are fulfilling.

Envisioned Interaction

Lavanya is a developer at a small startup that is creating a new real-time strategy game. She has just had an idea for a

CHI 2015, April 18–23 2015, Seoul, Republic of Korea

ACM 978-1-4503-3145-6/15/04 $15.00
http://dx.doi.org/10.1145/2702123.2702565

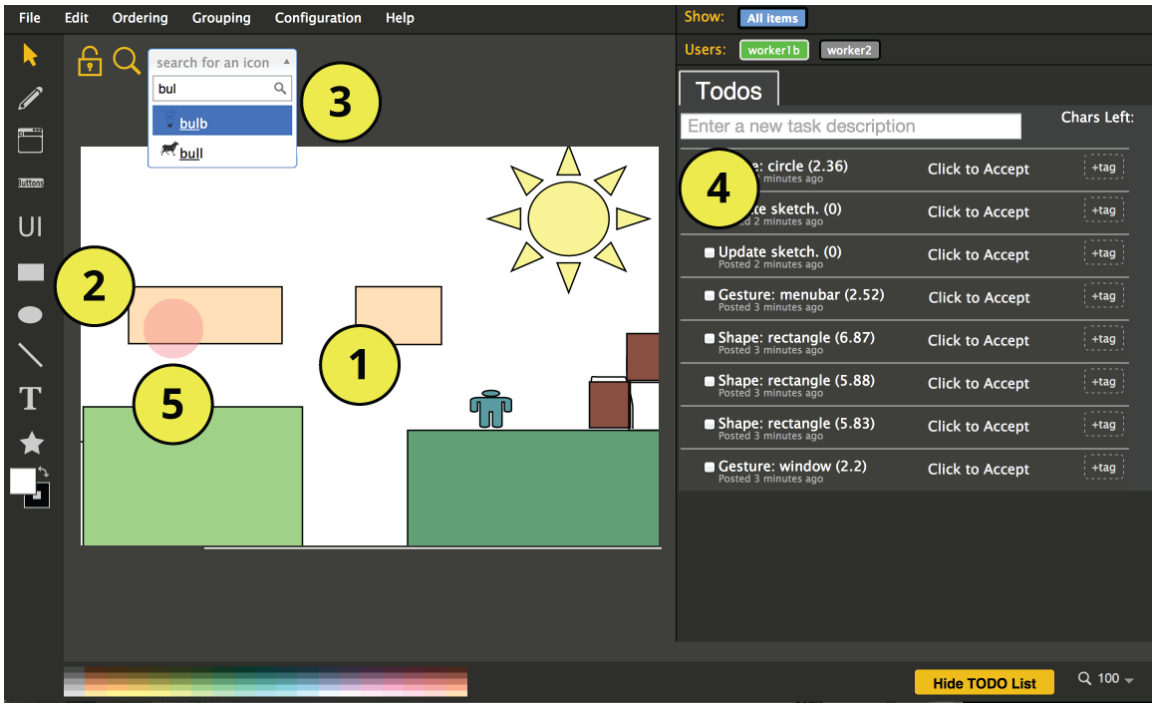

Figure 1. Apparition allows designers to quickly create functional interface prototypes using sketches and verbal descriptions. (1) A collaborative canvas where the user(s) and workers can draw. (2) Drawing tools and icons to quicken workers' creation of UI elements. (3) A search function to help workers find relevant icons. (4) A to-do list that shows what has been drawn by the user but not yet converted to UI elements. Workers can "accept" tasks to signal what they are currently working on. (5) "In-progress" markers for workers to show where they are currently working to avoid conflicts.

new enemy character. Lavanya flags her coworker Ellen to tell her about the idea, but Ellen is unsure whether the character will fit the feel of the game. In response, Lavanya opens Apparition and begins to quickly sketch out a scenario.

As Lavanya sketches a version of the game UI, her sketches are immediately converted into properly aligned interface elements. In less than a minute, Lavanya has sketched a rough version of the game's UI, complete with a handful of enemy characters. "See, these creatures run away when I get too close," Lavanya explains, "they find some distance and then start following me again." As she describes her idea in more detail, the new enemy characters on screen begin to look and act as she describes. Ellen then pulls out her tablet, connects to the same Apparition session as Lavanya, draws a few of her own characters, and sees how the system responds in various situations. When Ellen finds situations that were underspecified, Lavanya describes aloud how the approach would work, and the system-controlled characters change their behavior to match. Impressed, Ellen tells Lavanya she should demo this approach at the team meeting later that afternoon.

While Lavanya prepares her talk, she puts Apparition into "Bake In" mode, which continues to improve the prototype without further end-user intervention. A few hours later, when demoing the system to the team, Lavanya notes that the workers have replaced the early images with more refined assets. The previously described behaviors are still in effect but are continuously refined as people ask questions and make suggestions. Team members can copy the interface and branch, comparing ideas side-by-side.

Outline and Contributions

Apparition works to make near-instant prototyping possible by using human computation to power interactions that automated systems cannot yet handle alone. Apparition users can fluidly prototype designs by sketching and speaking on a shared canvas. If its automatic sketch recognition algorithm recognizes UI elements, Apparition replaces them with higher-fidelity representations [14]. For all other elements — friend lists, game characters, map interfaces — paid crowd workers coordinate the transformation of each sketch into the desired prototype element. Given more time to "bake in," the prototype visuals continue to develop. Users can speak desired behaviors, *e.g.*, "when I click the button, the friend list slides over and the map pops up," and crowd workers Wizard-of-Oz the behavior the next time the action is triggered in real-time.

We make the following contributions in this paper:

- The first crowdsourcing system where workers self-coordinate to complete a complex task in real-time.
- The concept of intelligent prototyping tools powered by an on-demand, self-coordinating crowd.
- Methods and tools for coordinating large groups of people on simultaneous control and creation tasks.
- Apparition, a tool that lets crowds of online human workers collectively help users create working interface prototypes from sketches and verbal descriptions in seconds.
- Validation of our approach through examples of the types of prototyping that Apparition makes possible.

We begin with a discussion of the background and related work in design, prototyping, and crowdsourcing. We then

discuss how Apparition works and demonstrate that it can quickly create accurate, functional prototypes. We conclude with a discussion of our results and future directions of research made possible by releasing Apparition.

BACKGROUND

Apparition builds on 20 years of Wizard-of-Oz prototyping tools and demonstrates how paid crowds can form a scalable, always-available set of wizards. It also extends the literature from static screenshots to functional prototypes with dynamic, intelligent behaviors. In this section, we discuss prior work in UI sketching, prototyping, and crowdsourcing.

Easier Prototyping with Sketching and Wizard-of-Oz

Tools such as Balsamiq (www.balsamiq.com) or JustInMind (www.justinmind.com) allow designers to create interfaces without requiring the same significant intervention during testing that paper prototypes do. However, these applications require training and a large amount of effort to create, making them impractical during the earliest stages of design [14].

Systems such as SILK [14] and DENIM [20] reduce the overhead of prototyping by recognizing designers' sketches as interface elements. Designers can then create a linked storyboard defining appropriate state transitions for different interactions. While this is effective, the scope of elements they recognize is limited and interactions can only result in individual state updates.

d.tools [9] allows designers to begin with a rapid prototype created from visual statecharts and move towards implementation versions by adding code to parts of the statechart. d.note[8] allows users to revise both appearance and behavior by annotating screenshots and storyboards. Apparition allows similar revisions of behaviors and appearance in real-time during interaction via speech and sketching.

The advantages of multi-modal interaction, *e.g.*, combining sketching with spoken natural language descriptions, have been studied in diverse settings and contexts [3]. Natural language provides a means of capturing many types of information that is difficult or unnatural to capture in sketches, *e.g.*, temporal relationships, behaviors, and spatially disjoint elements [3]. By integrating voice, these selection, scoping, and description issues become single-interaction tasks rather than multi-step processes. Apparition uses multimodality to expedite the creation process for the end-user and clarify meaning for crowd workers. We aim for a process like describing a system to a co-located colleague.

Designing behaviors is difficult partly because of limitations in tools available to designers—a vast majority of 267 interviewed designers found programming behaviors more difficult than visual design [26]. More importantly, behaviors targeted by these designers were too diverse to embed into any one system with a fixed set of supported features. Apparition addresses this problem by leveraging human intelligence with spoken language to support highly diverse sets of behaviors with just a simple description. These designers also struggled to fully communicate their desired behaviors with the developers building the system. One goal of Apparition is to allow designers to easily create a "living spec" that developers interact with to better understand the designers' intended behaviors.

Wizard-of-Oz control simulates complex functionality in prototypes [11] – a human "wizard" provides feedback or control to simulate difficult-to-implement features. For instance, to test whether a chat-based support tool will help users better find information on a webpage, an instant messenger tool might be built into the site so that users can communicate with a designer instead of an automated system.

Tools such as SketchWizard [5] make Wizard-of-Oz prototyping easier by allowing the end-user to create pen sketches, and a wizard behind the scene converts their sketch into geometric shapes and elements. Because of the speed and accuracy needed to keep up with even within-lab use cases, situational planning and practice is needed for wizards to support trials with users in real-time. In contrast, Apparition provides a canvas and coordination tools that allow many crowd workers to share the work of keeping up with user interactions in live sessions and demonstrations. Thus, Apparition provides open-ended support without prior planning or setup.

Apparition's ability to use speech also extends the expressiveness of the interfaces and behaviors that can be prototyped. Turvy [24] used Wizard-of-Oz to control an intelligent agent that users could instruct in natural language. Turvy let researchers see how users would really interact with their agent, leading to insights that can aid in the development and automation of a released system. Prior work has also explored Wizard-of-Oz conversational interaction, including combining the input of human wizards and machines [7, 18, 27].

While quicker and easier than a full implementation, these Wizard-of-Oz powered systems are generally one-off and can still require significant development effort. Apparition overcomes that by providing a set of broadly-applicable tools and techniques for coordinating multiple workers so that complex, concurrent actions are supported with relative ease.

Crowdsourcing and Real-Time Human Computation

Human computation engages people as part of a process to solve problems that automated approaches cannot handle alone. Crowdsourcing, where an open call is made to a population to elicit responses, effectively accesses human intelligence for computational tasks. Workers can be recruited on demand from platforms such as Amazon Mechanical Turk and engaged for short time periods to complete a task. However, this process has the caveat that workers are unknown to task requesters and might be of low quality or even malicious in rare cases.

Most work has focused on coordinating workers by dividing work into small microtasks [21] or well-specified ongoing roles [16]. Prior work has also explored how crowds themselves might be used to recursively generate these workflows [13] with some end-user oversight. In constrast, Apparition must be able to create new roles and tasks as soon as the user describes an intent, so real-time coordination for defining and adopting these roles is our goal.

Real-Time Crowdsourcing

Real-time human computation has been explored in systems like VizWiz [2], which showed that the crowd could answer visual questions in less than 30 seconds, and Adrenaline [1], which formalized the retainer model for bringing a group of crowd workers together in two seconds. Systems like Legion [17] have explored how a synchronous crowd can work together effectively once recruited (in that case to collectively control existing user interfaces). Apparition allows the end-user to work with the crowd to produce an artifact.

Chorus [18] engaged a group of workers in a conversational interaction with a user, using an incentive mechanism and memory space to encourage the crowd to act as a single consistent individual. Workers powering Apparition must also be consistent because prototypes must reliably function in the same way over multiple interactions with end-users.

Crowdsourcing for Drawing and Rapid Evaluation

Crowdsourcing inherently provides access to individuals with diverse skills, which provides guidance to users during the design process [22] and improves specific skills like drawing. Limpaechter *et al.* collected sketch data from a crowd of players of their DrawAFriend game and used the resulting model to correct user-drawn lines in real-time [19].

Foundry [23] allows end-users to create "flash teams", computer-mediated teams of expert crowd workers that were able to complete complex tasks such as design prototyping and animation that are produced by experts, rather than microtask workers, from the crowd.

Glance [15] uses the crowd to help users explore video datasets by quickly marking arbitrary user-specified events. This divide-and-conquer process is common in crowdsourcing because it allows the inherent parallelism of the crowd to help solve problems faster. Apparition uses a similar division of labor to make complex, parallel control tasks possible, but unlike prior work in crowdsourcing, it supports worker self-coordination and task-finding in real-time.

While existing Wizard-of-Oz approaches rely on a single user who is an expert with a particular system, crowdsourcing has the potential to provide an on-demand workforce that can be hired for exactly the amount of time needed to power an interaction. Crowds also have the potential for supporting more complex, parallel actions than single controllers could.

APPARITION

Apparition's goal is to let designers create prototypes, complete with visual elements and behaviors, in real-time as they freely sketch and describe their ideas. To do this, Apparition connects the designer to a crowd of paid microtask workers who collectively translate the designer's intentions into a functional prototype in real time.

The primary challenges are:

- supporting synchronous editing of interface mockups
- avoiding repetitive work and production blocking
- allowing the crowd to upgrade the interface over time
- remembering behaviors so that the interfaces are immmediately Wizard-of-Oz'ed

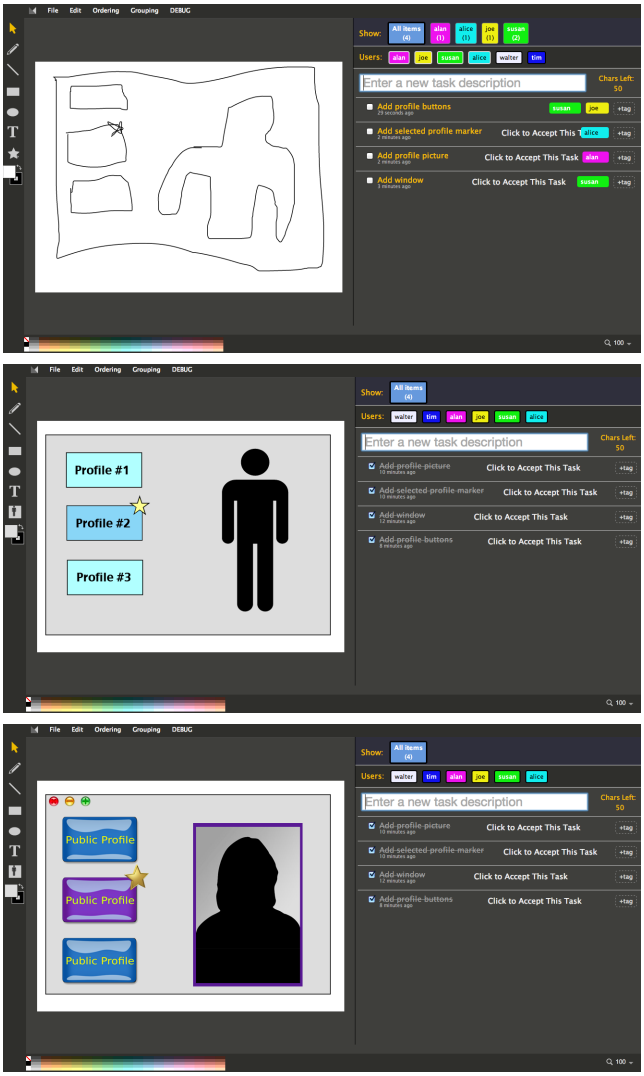

Figure 2. Designers roughly sketch their interface while describing it in natural language (top). As they do, crowd workers collaboratively update the interface into a low-fidelity prototype (middle). Over time, the crowd workers further improve its quality (bottom). As these interfaces are created, they are immediately functional, even when adding complex behaviors via collective Wizard-of-Oz control by the crowd.

System Architecture Design

Apparition can be run from any device with a web browser, making it possible to generate interfaces in settings where generating high-fidelity UIs is currently impossible, such as from a mobile device. Audio is streamed from the user to each crowd worker using Adobe Media Server. Users can pause the audio stream at any point they wish (for privacy).

Designers begin by sketching a low-fidelity version of their interface using either a pen and tablet or a mouse. Automatic gesture recognition identifies user strokes when possible (similar to SILK's approach [14]). This is done using a specially trained $1 Recognizer [29]. If the recognizer cannot confidently classify and convert the sketch into the UI element it represents, Apparition asks workers to produce a higher fidelity rendering in real-time by using simple visual prototyping tools provided to them.

To bring the interface to life, behaviors can be animated by moving, adding, and removing elements on the canvas. Natural language descriptions help the crowd fill in details of both the interface layout and behaviors despite minimal content in the initial sketch. Thus, designers can quickly iterate on their ideas without having to sketch each version in detail.

While beyond the scope of this paper, it is worth noting that our approach can even work without the user directly sketching their idea on real media (*e.g.*, paper or whiteboard). From a verbal description or gestures viewed via a webcam to help ground the layout, workers can still get a rough idea of what the designer intends and render it.

Below, we describe operational aspects of Apparition and discuss features that allow workers to coordinate effectively to create prototypes within seconds of when users initially sketch them. These design decisions were made over the course of building Apparition during informal evaluation sessions with dozens of in-person volunteers, workers from Mechanical Turk and oDesk, and other contributors.

Canvas

Apparition's canvas is a version of SVG-Edit (code.google.com/p/svg-edit), a browser-based vector graphic editor, that we modified to support multiple users. Our synchronized version uses Meteor (www.meteor.com), a JavaScript framework that allows us to quickly share information between clients. Each worker connects to an active session and can see the actions of others as they work. The core tool is similar to Google Draw, but it includes custom features making it possible to coordinate groups of workers creating, editing, and animating simultaneously — something Google Draw was not designed to do.

Apparition improves efficiency of workers with a library of standard interface elements. The most common of these are provided as top-level elements in a toolbar (seen on the left-hand side of Figure 1). For the larger set of less frequently used elements, we provide a search tool suggesting UI elements based on partial, related, or non-exact element names, for example "profile", "tab", or "star." Search helps prevent workers from spending too long browsing for an element that does not actually exist.

Roles

Apparition synchronizes canvases so that designer(s) and crowd workers always see the current design. Roles can also be defined for different user groups. The information shown and synchronized can be modified to best fit the needs of each type of user. For example, designers may not need to see worker to-do items (discussed later) or workers' in-progress elements, so these are synchronized after completion. Conversely, a user's input is shown immediately to workers.

Workers can also see who is online, and each worker has a globally consistent and unique color assigned to them. This color code applies to all interactions that a worker has with the system, so other users can get a sense of what actions are being performed by a single individual. This helps with semantic grouping and understanding what tasks others are performing — key elements in maintaining coordination.

ENABLING REAL-TIME WORKER COORDINATION

No one worker can keep up with a designer who is sketching and describing interfaces in real time. Once a worker has converted each sketch element or behavior task—each of which can take several seconds—the designer has long since moved on. For people, understanding is not typically the roadblock in this setting, but rather creating the grounded representation of the described concepts quickly enough. In early versions of Apparition, workers fell prey to diffusion of responsibility [4], and all would wait for someone else to pick up each task. Making Apparition responsive required a method for coordinating workers around very open-ended, potentially subjective tasks such that they do not conflict or repeat work.

To solve this problem, Apparition uses a write-lock coordination mechanism that allows workers to self-manage task delegation in real time.

Naive coordination fails

To coordinate workers and ensure that no task is missed, even when the designer is speaking quickly, Apparition uses a to-do list that synchronizes short messages from the system and workers to other workers. The intention is for this to-do list to populate automatically as the designer works. Using a partially-automated to-do list prevents the latency issues associated with completely manual curation of this set and any resulting duplicated work.

Apparition automatically groups nearby sets of sketches into a single unit and assigns them as a to-do item in the list. By mousing over the to-do item, workers can see which canvas elements are being referenced. Workers can claim to-dos and add new ones if desired. In addition, sketches that the gesture recognizer captures with only medium confidence are automatically converted into to-do entries. This approach strikes a balance between the benefits of automatic guessing and the risks of introducing errors.

Unfortunately, our early observations were that this approach produced significant conflict. Workers focused on their self-defined tasks and often missed looking at the to-do list on the side of the screen at a critical time. This resulted in blocking actions where multiple workers would try to complete the same task — in the worst case, accidental conflicts caused work to be undone.

Ad hoc leaders with delegation power can be extremely effective for short-lived teams [12]. Thus, we tried specifying some workers as "managers" who could create tasks and assign others to tasks. However, requiring that a worker first tag and route a task introduced unacceptable latency.

Write locks enable effective distributed coordination

The to-do list was insufficient because workers placed their primary attention on the actual canvas. In other words, it was not a problem of the input jobs, but a problem with write-locking the actual "data".

To overcome this, we added a form of informal write-locking where workers can click a location and drop an "in-progress" marker that tells other workers that they are working in that general vicinity (Figure 3). These markers do not actually

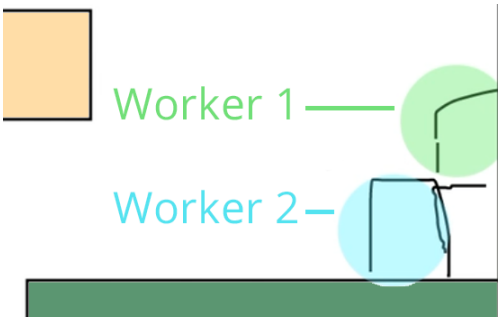

Figure 3. An example of Mechanical Turk workers using "in-progress" markers in a live trial. Each worker's marker is a different color (here, green for worker 1 and blue for worker 2) translucent to interface elements and editing actions and can be added in two clicks. By making it easy for workers to signal to others what they are working on, production blocks and repeated work are avoided.

prevent any edits within the designated region, but they allow a worker to signal to others that they are working on a task associated with this area. Markers can be tied to specific to-do entries, or they can be used in place of them when no text is needed to describe the task that they are performing based on context. At most one of these markers is visible for any worker at a time, and placing a new one automatically removes the prior one. While they are displayed on top of all canvas elements, markers are partially transparent and completely un-selectable, preventing them from blocking others who are currently editing canvas content.

Because we typically expect fewer than 10 workers to be present simultaneously, these markers also do not clutter the interface as it is being edited. They also do not appear to end-users, who only see the results generated by workers, not the coordination elements they use.

Using these markers, workers self-manage and implicitly or explicitly claim elements of the to-do list. This increases parallelism and reduces conflicts.

Supporting Concurrent Editing: Replace-at-Layer

One significant hurdle to concurrent editing in small regions of the document arises from element layering. The most common practice of visual design tools with respect to layering is to have the layer index of an element be determined by the temporal ordering of creation (as seen in Adobe Photoshop, Google Draw, etc.). This means that if a window is sketched and then a button is drawn, the button will be one layer above the window. This is a natural interaction in single-user settings — but when a crowd of workers is simultaneously editing a single region of the canvas, replacing a window sketch by adding a window element to the canvas results in the window element arriving as the top-most selectable and visible item, covering up any other elements that other workers might be attending to. This leads to a production block for everyone working in that area, and in practice, often leads to errors being introduced by workers performing actions not meant for the now-topmost window.

To overcome this, we added a *replace-at-layer* feature that lets workers select a sketched figure, create a replacement element, and then replace the sketch with the new element at the same layer index in a single operation. This means that workers can create elements behind others in a single click.

To prevent mistaken operations, replace-at-layer first checks for spacial proximity to see if the new element overlaps with the same approximate region (in our case 150% of the size of the original sketch). We also require the worker to explicitly select the element they wish to replace first, as a separate selection action, so that the automatic selection of an element after initial creation will not be used to trigger a replacement (so that sequential creation operations can be done seamlessly). An optional replacement confirmation message can also be enabled if a safety check is desired. Once we have determined that the worker intends to replace a given element, we find that element's location in the display stack, remove the initial sketched element, and then add the worker's newly-created element at the same relative location (above and below the same content as the original sketch).

Protecting Against Malicious Workers

During initial trials, we saw that reliability might be a concern, as it often is with crowdsourcing tasks. We ran five trials with three workers in each. In three of five trials, one worker moved elements out of place, negatively impacting the final result. In two of these cases, that worker was the last one to edit the prototype.

To address potential worker confusion, we added tutorials and examples. To flag malicious or contrarian workers from disproportionately impacting a task, we looked at the history of workers' operations. If a worker is performing a specific action repeatedly over time, the system flags them as potentially malicious. For example, a user removing many elements created by others (where no other worker finds that those elements need to be removed) might be deleting useful elements. On the other hand, a worker creating a large set of elements when others are not might be spamming.

INCREASING FIDELITY AND ADDING BEHAVIORS

In this section, we explore how Apparition can move beyond quick mock-ups and static interfaces by making these interfaces improve themselves over time and come alive.

From Mock-Up to Polished Prototype

Apparition's real-time nature means that the level of detail of the immediately-available interface is limited — we don't want workers to spend extensive amounts of time worrying over small details. However, over time, workers can refine elements to create a more polished version. This additional time might arise when the user leaves for a period of time, such as for lunch, overnight, or over a weekend. As such, we assume that this additional interface refinement can operate as an asynchronous offline process.

To draw on prior work that has shown parallel prototyping leads to more creativity [6], we use a branch-and-merge algorithm similar to Multiverse [25]. When users set the system to "bake in" mode, they select time and cost bounds and Apparition automatically begins to create parallel copies (typically 3-5 per round) of the current interface. Each of these copies is distributed to multiple crowd workers who work to

improve the existing prototype either synchronously or asynchronously (in our experiments, we allow for either depending on workers' arrival time). For context, workers are also able to access the original user descriptions of the interface (via audio and screen recording).

Once several different workers (typically 5-8) have contributed to each prototype, the final results of each are shown to a new group of workers to select the best version. The winning version is then copied into multiple universes for the next iteration and those are again redistributed to new workers. This iterative process continues as many times as the time and cost bounds will permit. At the end, the result is a single "best" refinement of the original mock-up. For example, in Figure 2, we can see the initial sketch in the top frame, the immediately-available version in the middle, and a more polished version that was created a few moments later, after workers had time to find graphic resources on the web.

Prototyping Interactive Behaviors

We have described how Apparition allows designers to quickly see an improved version of the interface they describe, and given more time, see an even higher-fidelity version if they choose. But interfaces are not static artifacts.

Apparition allows designers to describe the desired behavior of their prototype out loud, and then immediately interact with a working version of their interface. Workers hear the description of the correct action and then can immediately Wizard-of-Oz the behavior as needed. To modify or correct results, the end-user can simply describe them verbally and click if direct reference to an object is needed.

The to-do list can again be used by workers to coordinate roles in supporting repeated interaction. For simple behaviors, such as popping up a window, workers might be able to accept multiple tasks, while for more complex behaviors, such as controlling a character in a video game so that it hides from the player as they move, a worker might need to focus on only that task. Once a worker has assigned themselves to a task, they can continue to monitor it until they choose to leave or hand off responsibility.

Showing User Interactions

When describing to-be-supported actions in natural language, users often will use language that requires supporting context to understand [10]. For example, if a user says "When I click *here*, the value of the counter will increase," the position referenced by "here" must be specified. While this is done naturally in communication between co-located individuals, Apparition supports this using a shared visualization of user interaction on the canvas. When a user clicks a position on the canvas, the point where they clicked is displayed as a dot animation to all workers. The animation lasts $\sim$2 seconds, but when paired with speech, it provides a clear signal to workers. Support for swipes and other gestures using line animations can also be added. These gestures support not only referential phrases, but also interactions with the interface that are not mentioned in speech directly.

Adding Constraints

While Apparition's prototypes are meant to be dynamic, users might want to specify that some sub-component of their system is complete, while wanting others to be improved or iterated on. Apparition lets users specify elements which should be "locked" in place as immutable either at the beginning of any round during the UI refinement process or during editing in a live setting where animations might otherwise move a given element. When a canvas-wide lock mode is toggled, these elements are made immutable for both workers and users to prevent these locked elements from being modified during interactions.

However, freezing an element in place is not always the right constraint. In some situations, users might wish for an element to simply remain in a certain region or have a certain range of motion during animation. For example, a user may want to bound a map pin to stay within the map. To allow for this, users can define bounding boxes around elements that restrict the motion of that element to a specific region when lock mode is activated.

In combination, locks and bounding boxes help workers create accurate animations and effects with minimal effort. For example, to create an accurate slider animation, a line or bar can be locked in place, and a handle element can be restricted to a narrow horizontal region on top of that line (Figure 4). Then, when the canvas is locked, workers are only able to move the handle of the slider along the axis, and only within the bounds of the corresponding line element. This avoids jitter and makes the animating worker's task simple.

EXPERIMENTS

To verify Apparition's ability to create interface prototypes in real time, we first selected a set of five common interface types to mock up and had a UX designer create napkin sketches of each as a starting point. We added appropriate behaviors to each, resulting in the following data set:

- **Map viewer**: Contains – zoom bar, divider bar, main window, and selector buttons (x3). Behaviors – zoom in (x3), zoom out (x3), click to change window contents (x4).
- **Profile manager**: Contains – divider bar (x2), main window, and selector buttons (x6). Behaviors – zoom in (x3), zoom out (x3), click to change window contents (x4).
- **Document editor**: Contains – main window, option buttons (x2), back arrow, and selector buttons (x5). Behaviors – button click to change window contents (x3), button click to change button color (x6).
- **Platformer game (*e.g.*, Super Mario Bros.)**: Contains – game character, separated ground (x2), large floating block, small floating block, stacked boxes (x3), and sun. Behaviors – move to clicked location (x4).
- **Real-time strategy game**: Contains – player character, enemy characters (x3), radar, control menus (x2), divider bar, menu button. Behaviors – follow player character with 3 enemies (x2), follow player character with 3 enemies (x3).

We recruited 46 workers from Amazon Mechanical Turk using LegionTools [15], a tool for quickly recruiting and directing synchronous crowds of workers. We ran 10 trials, two for

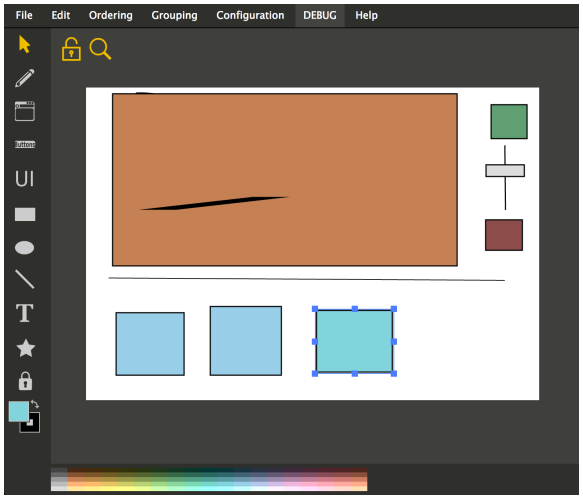

Figure 4. An example of an interface being created during one of our trials. End-user input consisted exclusively of a pencil tool and natural language speech, which workers converted to UI elements.

each of our five example interfaces (with 23 workers in each half). Trials were moderated by a researcher who acted as the end-user and tried to re-create the mockups we obtained from the UX designer. Each trial lasted approximately 5 minutes for workers, including the tutorial, and paid $0.95 ($19/hour). We limited our tasks to U.S.-based workers with an approval rate of at least 90%. We recruited 3-6 synchronous workers to convert the interface that they were shown. The task consisted of both a *creation and editing* phase, as well as a *behaviors and actions* phase. In half of the trials, workers did not have access to "in-progress" markers, while in the rest they did.

We measure precision using the number of actions taken that were intended by the requester and recall using the number of actions intended by the requester that the crowd completed. If, after a short while, the request had to be repeated or highlighted to workers, the first request was counted as a miss and the second question counted as a separate instance.

8 Seconds from Sketches to Interfaces

Creating and editing interface elements includes all steps taken by workers to convert sketches to real elements, create or remove elements from verbal cues, change an element's static position, resize elements, and change their color. These actions had a median latency of 8.3 seconds (mean 10.9s, $sd = 6.2$s), with 90.8% precision and 97.6% recall.

Workers React with Behaviors in 3 Seconds

Behaviors and animations include any element placement, coloration, dynamic movement, or visibility that is an effect of interacting directly with the interface (*e.g.*, clicking a button), rather than a direct user request to change the state of the interface. Performing these actions had a median latency of just 3 seconds (mean 3.6s, $sd = 2.3$s), and achieved 92.9% precision and 90.5% recall.

In-Progress Markers Prevented Production Blocking

We found that workers were very willing to use in-progress markers when they were available: 72.2% of all workers used markers during the live session, and they were used in all sessions in which they were available. An average of 7 markers were placed per trial (median 7.5s, $sd = 6.67$s), making the average per-worker usage 2.7 times per session.

Markers helped workers avoid redundant or conflicting work, which appeared to result in fewer scenarios where no worker completed a task because they were unsure of if others were already doing it. This resulted in a significant improvement from 82.3% recall in the trials without markers, to 97.0% ($p < .05$) in the trials with them. While marker use also resulted in higher precision (94.8% with markers versus 90.4% without them), this difference was not significant ($p > .5$). Latency was also not detectably impacted ($p > .8$).

Markers were almost exclusively used for creation tasks, specifically when the creation task was grounded by a sketch. This means that it overlaps significantly with the use case of the to-do list. However, as expected, we observed that the usage of the to-do list was almost entirely replaced by markers in three of our five trials with markers. In only one case was a marker used to denote ownership over a behavior.

DISCUSSION

Our evaluation suggests that Apparition can keep pace with designers, transforming each sketch element within eight seconds into a higher-fidelity representation. Once created, the prototype quickly reacts to user input. The result: a user could step up to a tablet, sketch a game of Super Mario Brothers, and start playing it within seconds.

We envision designers receiving feedback far more quickly than they can using methods like paper prototypes. Apparition allows iteration within a brainstorming session or meeting and allows users to immediately see how their changes would affect the interface, reducing the need to speculate. This may result in more helpful feedback from early users. The crowd itself can also provide feedback. Prior work has explored how individuals can get feedback from crowd workers during the design process [22].

Apparition can be used with non-microtask crowds as well. For example, in a design meeting, designers themselves may act as crowd workers to help flesh out their own or others' ideas and view the resulting interactions. More reliable expert workers can also be recruited from contract-work sites, such as oDesk or Elance. These workers can be especially helpful when producing higher-fidelity versions of the desired interface. These workers can include graphic artists and designers, as well as programmers who implement functionality that the end-user designer did not but that still requires only an amount of effort reasonable for the targeted fidelity.

Deleting Content

One of our concerns when designing Apparition was that malicious or confused workers might delete parts of the interface while the user was designing. To prevent this, we designed a tool to detect unilateral actions such as deletes and catch potentially-malicious workers. However, our trials encountered no examples of malicious workers deleting valid content. Instead, there were two instances where a worker

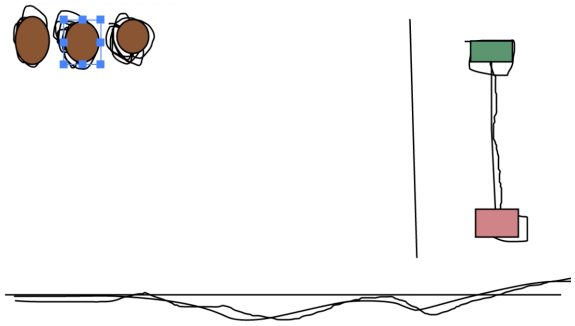

Figure 5. In some of our trials, workers hesitated to delete content. This resulted in valid elements being drawn on top of, but not replacing, user sketches (possibly because workers were not comfortable deleting what the user created). Future versions of the system will automate approaches to replacement.

mistakenly added content. From the usage, both cases appeared to be caused by accidentally selecting a tool, and then trying to select content (resulting in an element created along the intended drag path).

Interestingly, we found that workers' hesitation to delete content caused many more issues. In extreme cases (Figure 5), these remaining sketches detract from the visual aesthetic of the prototype and create problems with motion-based behaviors by leaving a sketched copy of the element in its original location, adding to clutter.

Our element replacement tool was meant to overcome some of this. While it helped in several settings, it was calibrated to work only when the system was extremely confident in the placemen. Even then, it prompted for confirmation. In future versions, we will slightly loosen these restrictions and explore methods for intelligently determining appropriate replacement situations based on the locations and sizes of the elements, input from the gesture recognizer.

Remembering Interface Behaviors

As part of our UI behavior and animation experiments, we asked workers to perform simple actions based on user input. Because these actions are defined in natural language and not repeatable, workers are required to remember what behavior was requested. For instance, if told that a certain action should be performed in response to a certain button being clicked, the crowd is expected to remember the association. To test how well the crowd can collectively remember behaviors, we conducted a small set of tests that asked the crowd to perform a behavior that was requested K interactions ago.

In our sample of 16 one-, two-, and "many"-ago interactions (with 3, 8, and 5 examples respectively), we found that the crowd was able to recall the correct behaviors and apply it in 81% of cases. Interestingly, the "many" cases were all performed correctly. This suggests that the crowd can reasonably recall behaviors that users describe, though longer-term memory will require methods for letting the crowd store information for future use either by themselves or others.

Communication

One common piece of feedback that we got from workers was that they wanted to chat with the requester and other workers. Providing this feedback and between-worker communication channel will likely help workers better understand the task early on but runs the risk of distracting workers and slowing down response times [18]. Some workers were initially confused by the task, but others found the task to be easy to understand given prior experience with graphical editing tools. Over time, workers without this experience may learn how to complete our tasks. Nevertheless, this experience is common enough to be useful among workers on Mechanical Turk.

Limitations

Our study focused on demonstrating that the collaboration environment that Apparition provides allows crowd workers to create prototypes within seconds. Our study did not use external design participants as end-users so that we could control the settings and designs more carefully. In the future, our goal is to release Apparition, allowing us to explore how real-world use of intelligent prototyping systems can impact the design process.

FUTURE WORK

Our results with Apparition demonstrate that it is possible to supply synchronous crowds with relatively light-weight coordination tools and enable them to create interfaces and behaviors within seconds of their original description. Future improvements will aim to make the process appear more seamless to end-users by letting crowd workers prepare changes before making them visible, while preventing redundant work by leveraging the fact that workers actively use markers.

Programming Crowd Functions

Complex behaviors can also rely on a series of descriptions given by a user. However, we assume that the crowd is dynamic and that workers come and go — no one worker can be relied upon to be available for a fixed span of time. To ensure that behaviors are retained even as the crowd changes, we need workers to be able to record the instructions for how to respond to input. Capturing this knowledge can be thought of as 'programming' a crowd function – it defines an atomic task that can be completed by any worker who arrives in the future. Currently, the to-do task description serves this role, but ongoing work will allow workers richer ways to pass on knowledge. For instance, they may annotate the canvas with animation paths or even point to recordings of past responses.

Automatically Capturing Behaviors

Many behaviors are associated with actions that are repeated over time. We want to be able to not only allow the crowd to simulate these behaviors, but also to learn when these behaviors should occur. The simplest way to do this is to identify how a crowd worker responded to a user's input (such as selection), and ask them if this should always be the response. If so, then we can record the set of actions they take in response by bounding the sequence by the user input in the beginning and the worker marking the to-do item complete at the end.

Ongoing work will allow Apparition to replay action sequences automatically when the appropriate input is received. Workers and designers can mark this automatic action as incorrect if a situation arises where it is not appropriate. We

will explore automatically finding the differences in the state of the interface so that we can intelligently decide when to execute recorded sequences.

Transitioning to Code

We are also exploring ways to add code directly. Since we know the element type, we can begin to automatically add Javascript event listeners, allowing designers to add real code with low overhead. For example, if a designer edits a button element, we will show a pop-up window that lets them enter code for a 'click' event handler. Elements can also be easily referenced from code. This allows basic functions to be implemented on-the-fly as desired, while more complex functions are handled by the crowd.

Application developers can then use the partially crowd-powered version of the system as a "living specification" when creating production code. The existing code can be used as a reference, and the notes that crowd workers have made about behaviors can help clarify issues that might be otherwise missed from a traditional paper writeup. Finally, developers can use the crowd-powered version of the interface to answer questions about use cases that might have been otherwise missed or unclear from the documentation.

CONCLUSION

We have presented Apparition, a system that allows designers to create working interface prototypes in real-time by sketching and describing its functionality in natural language. The prototypes created by Apparition are over 90% accurate to the user's intent, and the system can respond to create elements within 8 seconds and prototype behaviors within 3 seconds of the user describing it. Apparition will allow designers to create prototypes fast enough to iterate within single design sessions, improving the feedback they receive.

ACKNOWLEDGEMENTS

The authors would like to thank Geza Kovacs and Patrick Baudisch for inspiring this idea. This work was funded by National Science Foundation Awards #IIS-1149709, #IIS-1218209, #IIS-1351131, Google, an Alfred P. Sloan Foundation Fellowship, and a Microsoft Research Ph.D. Fellowship.

REFERENCES

1. Bernstein, M. S., Brandt, J. R., Miller, R. C., and Karger, D. R. Crowds in two seconds: Enabling realtime crowd-powered interfaces. UIST 2011.
2. Bigham, J. P., Jayant, C., Ji, H., Little, G., Miller, A., Miller, R. C., Miller, R., Tatarowicz, A., White, B., White, S., and Yeh, T. Vizwiz: nearly real-time answers to visual questions. UIST 2010.
3. Cohen, P. R. The role of natural language in a multimodal interface. UIST 1992.
4. Darley, J. and Latane, B. Bystander intervention in emergencies: diffusion of responsibility. J. Pers. Soc. Psych. 8(4) (1968): 377.
5. Davis, R. C., Saponas, T. S., Shilman, M., and Landay, J. A. Sketchwizard: Wizard of oz prototyping of pen-based user interfaces. UIST 2007.
6. Dow, S. P., Glassco, A., Kass, J., Schwarz, M., Schwartz, D. L., and Klemmer, S. R. Parallel prototyping leads to better design results, more divergence, and increased self-efficacy. *ACM Trans. Comput.-Hum. Interact. 17*, 4 (2010), 18:1–18:24.
7. Dow, S. P., Mehta, M., MacIntyre, B., and Mateas, M. Eliza meets the wizard-of-oz: Blending machine and human control of embodied characters. CHI 2010.
8. Hartmann, B., Follmer, S., Ricciardi, A., Cardenas, T., and Klemmer, S. R. D.note: Revising user interfaces through change tracking, annotations, and alternatives. CHI 2010.
9. Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. UIST 2006.
10. Hayes, P. J. Using a knowledge base to drive an expert system interface with a natural language component. *Expert systems: the user interface* (1988), 153–182.
11. Kelley, J. F. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst. 2*, 1 (1984), 26–41.
12. Klein, K., Ziegart, J., Knight, A., Xiao, Y. Dynamic Delegation: Shared, Hierarchical, and Deindividualized Leadership in Extreme Action Teams. Admin. Sci. 2006.
13. Kulkarni, A., Can, M., Hartmann, B. Collaboratively Crowdsourcing Workflows with Turkomatic. CSCW 2012,
14. Landay, J. A., and Myers, B. A. Interactive sketching for the early stages of user interface design. CHI 1995.
15. Lasecki, W. S., Gordon, M., Koutra, D., Jung, M. F., Dow, S. P., and Bigham, J. P. Glance: Rapidly coding behavioral video with the crowd. UIST 2014.
16. Lasecki, W. S., Miller, C. D., Sadilek, A., Abumoussa, A., Kushalnagar, R. and Bigham, J. P. Real-time Captioning by Groups of Non-Experts. UIST 2012.
17. Lasecki, W. S., Murray, K., White, S., Miller, R. C., and Bigham, J. P. Real-time crowd control of existing interfaces. UIST 2011.
18. Lasecki, W. S., Wesley, R., Nichols, J., Kulkarni, A., Allen, J. F., and Bigham, J. P. Chorus: A crowd-powered conversational assistant. UIST 2013.
19. Limpaecher, A., Feltman, N., Treuille, A., and Cohen, M. Real-time drawing assistance through crowdsourcing. *ACM Trans. Graph. 32*, 4 (2013), 54:1–54:8.
20. Lin, J., Newman, M. W., Hong, J. I., and Landay, J. A. Denim: Finding a tighter fit between tools and practice for web site design. CHI 2000.
21. Little, G., Chilton, L. B., Goldman, M., Miller, R.C. TurKit: human computation algorithms on mechanical turk. UIST 2010.
22. Luther, K., Tolentino, J., Wu, W., Pavel, A., Bailey, B., Agrawala, M., Hartmann, B., and Dow, S. P. Structuring, Aggregating, and Evaluating Crowdsourced Design Critique. CSCW 2015.
23. Retelny, D., Robaszkiewicz, S., To, A., Lasecki, W. S., Patel, J., Rahmati, N., Doshi, T., Valentine, M., and Bernstein, M. S. Expert Crowdsourcing with Flash Teams. UIST 2014.
24. Maulsby, D., Greenberg, S., and Mander, R. Prototyping an intelligent agent through wizard of oz. CHI 1993.
25. Murray, K. I. Multiverse: Crowd algorithms on existing interfaces. CHI EA 2013.
26. Myers, B., Park, S., Nakano, Y., Mueller, G., and Ko, A. How designers design and program interactive behaviors. VLHCC 2008.
27. Rossen, B., and Lok, B. A crowdsourcing method to develop virtual human conversational agents. *Int. J. Hum.-Comput. Stud. 70*, 4 (2012), 301–319.
28. Schon, D. The reflective practitioner. 1984. Basic Books.
29. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. UIST 2007.