Real-time Crowd Control of Existing Interfaces

Walter S. Lasecki[1]*, Kyle I. Murray*[1]*, Samuel White*[1]*, Robert C. Miller*[2]*, and Jeffrey P. Bigham*[1]

University of Rochester, Computer Science[1]
Rochester, NY 14627 USA
{wlasecki,jbigham}@cs.rochester.edu
{kyle.murray,samuel.white}@rochester.edu

MIT CSAIL[2]
Cambridge, MA 02139 USA
rcm@mit.edu

ABSTRACT

Crowdsourcing has been shown to be an effective approach for solving difficult problems, but current crowdsourcing systems suffer two main limitations: (i) tasks must be repackaged for proper display to crowd workers, which generally requires substantial one-off programming effort and support infrastructure, and (ii) crowd workers generally lack a tight feedback loop with their task. In this paper, we introduce Legion, a system that allows end users to easily capture existing GUIs and outsource them for collaborative, real-time control by the crowd. We present mediation strategies for integrating the input of multiple crowd workers in real-time, evaluate these mediation strategies across several applications, and further validate Legion by exploring the space of novel applications that it enables.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Human Factors, Experimentation

Keywords: real-time crowd control, real-time human computation, crowdsourcing, remote control

INTRODUCTION

Crowdsourcing has been shown to be effective at solving problems beyond the capabilities of current automated approaches [2, 3], but current crowdsourcing systems suffer two main limitations: (i) tasks must be repackaged for proper display to crowd workers, which generally requires substantial one-off programming effort and support infrastructure; and (ii) crowds generally participate asynchronously, without a tight feedback loop between workers and their task. This paper considers a new view of crowd computing that surpasses both limitations by using existing graphical user interfaces and putting the crowd in control of the mouse and keyboard. We introduce Legion, a system that allows end users to easily capture existing GUIs and outsource them for collaborative, real-time control by the crowd.

To use Legion, users first select a portion of their desktop interface that they would like the crowd to control, provide a natural language description of the task for the crowd to perform, and offer a price that they are willing to pay (Figure 2). Legion then forwards a video feed of the interface to the crowd and forwards key presses and mouse clicks made by the crowd back to the interface. To improve reliability, multiple workers are recruited to collaboratively complete the task. A fundamental question that we explore in this paper is how to effectively mediate crowd work to balance reliability with the desire for real-time control of the interface. Legion coordinates task completion by recruiting crowd workers, distributing the video feed, and providing a flexible mediation framework to synthesize the input of the workers.

Legion lets end users leverage crowdsourcing in ways previously not possible. Our original motivation was to provide a quick way of bootstrapping highly-robust, intelligent assistive robots. Such systems usually require significant (and costly) training to work automatically, are prone to errors, and so can often be controlled remotely by experts. We imagined a hybrid system in which robots could operate mostly automatically, but in which new tasks could be crowdsourced on demand for real-time control. Legion supports the flexible control of such existing remote-control interfaces.

We have used Legion to turn an inexpensive robot into one that intelligently follows natural language commands. We have outsourced bits of office work using a word processor or spreadsheet. We have used it to fill in for us while playing games requiring constant attention while we got a drink. We have used it to provide the intelligence of a predictive keyboard to make its suggestions quicker and more accurate. As we will highlight, not all of these use cases currently work flawlessly, but they illustrate the broad possibilities of outsourcing existing interfaces and motivate our work on real-time crowd control.

Legion supports experimentation with different ways of combining the input of multiple crowd workers in real-time while retaining reliability guarantees. Although there are numerous approaches that could be taken, we have implemented five *mediation strategies* in Legion that we compare in this paper: (i) control by a single crowd worker, (ii) mob rule in which all input from all workers is serialized and forwarded to the interface, (iii) voting over small time windows in which only the most popular vote is forwarded to the interface, (iv) dynamically choosing a random worker for control, switching only when they become inactive, and (v) using

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
UIST'11, October 16-19, 2011, Santa Barbara, CA, USA.
Copyright 2011 ACM 978-1-4503-0716-1/11/10...$10.00.

crowd-agreement to dynamically elect leaders whose input is immediately forwarded to the interface and whose time in control is a function of their reputation built over time. The most appropriate mediation strategy is context dependent, as we will demonstrate with experiments across several different types of applications and tasks.

Contributions

In summary, our contributions are the following:

- We articulate the idea of real-time crowd control of existing interfaces, and describe considerations for the design of applications in this space.
- We present a system, Legion, that lets end users easily outsource existing interfaces to the crowd and exposes a framework for mediating the inputs of crowd workers.
- We formulate several mediation strategies for aggregating the input of multiple crowd workers, and investigate these strategies in experiments with a diverse set of applications.
- We further validate Legion by showing several new types of applications that we created that illustrate interactive crowd assistance, programming by demonstration, and the mash-up of several desktop applications.

BACKGROUND

Individual users have controlled interfaces remotely as long as networked systems have existed, dating back to early terminals that allowed users to log in and control time-sharing systems. With graphical user interfaces came remote display protocols such as the X Window System [18] and Virtual Network Computing (VNC) [17] became popular. Remote control has also been used to compensate for limitations in mobile browsers. For instance, Highlight runs a full browser on its server, which is remote controlled by the mobile browser [14]. Specialized remote control systems even allow aircraft to be piloted remotely (Figure 1). The main difference between these prior systems and Legion is the idea that multiple users could collectively control the interface.

Real time groupware allows remote users to collaborate in shared online spaces [8], and many online games and multi-user dungeons (MUDs) likewise allow users to play or interact in the same space with one another. In contrast, Legion synthesizes the input of multiple workers to act as a single controller of existing interfaces. A few web-based games allow multiple users to control a single interface. For example, Massively Multiplayer Pong allows all of the current players to control the paddle [13]. Its interface displays both the "real" paddle position and the user-specific paddle position. Maynes-Aminzade *et al.* have brought these techniques into the real world by enabling large audiences to collectively control a projected interface with collective actions like leaning to the left or right [12].

In machine learning, meta-learners combine multiple *weak* learners for better performance [16]. A specific class of meta-learners called arbiters learn to combine the input of multiple base classifiers to arrive at a final decision in a supervised way and can work in an online fashion [4]. Legion uses the metric of crowd agreement that we have defined to learn how to combine crowd input in an unsupervised way.

Prior work has considered how graphical user interfaces could be controlled automatically. Early work in this area used operating system APIs, but these projects quickly ran into problems because limitations in the APIs meant that many interfaces could not be correctly interpreted and manipulated in this way. The CoScripter [11] web automation system leverages the openness of the web to reliably interpret and manipulate the web interface, affording the freedom to focus on high-level problems like end user programming and intelligent interfaces for interface automation. Recent projects have taken a more robust low-level, pixel-based approach to interpreting and manipulating GUI components [23, 7]. Legion crowdsources not only the interpretation and manipulation of GUI components but also higher-level planning, allowing greater flexibility in how end users decide to automate their interfaces and what can be automated.

Figure 1: Cockpit control center for a Predator UAV. A pilot and several sensor experts collaboratively control the aircraft remotely, although their effort is split between different functions. The unofficial role of the observers is to ensure that the pilot has everything that he needs so that he doesn't need to leave the controls. *Image courtesy of Bryan William Jones.*

Human computation was introduced to integrate people into computational processes to solve problems too difficult for computers to solve alone, but has not been applied to real-time control problems. Human computation has been shown useful in writing and editing [2], image description and interpretation [3, 22], and protein folding [6], among many other areas. Existing abstractions focus on obtaining quality work, and generally introduce redundancy and layering into tasks so that multiple workers contribute and verify results at each stage. Unfortunately, this takes time.

Most work in human computation has focused on finding appropriate abstractions for ensuring reliability for particular types of tasks – for instance, guaranteeing reliability through answer agreement [22] or the find-fix-verify pattern of Soylent [2]. These abstractions introduce redundancy and layering into tasks to ensure reliability, but this takes time and is not conducive for real-time control. Naive solutions like recruiting a single online worker may allow for real-time control, but would subvert existing methods of achieving reliability and are not robust to workers leaving (common in the crowd). As a result, new abstractions are necessary.

Several systems have explored how to make human computation interactive. As an example, VizWiz [3] answers visual

questions for blind people quickly. It uses quikTurkit to prequeue workers on Amazon's Mechanical Turk so that they will be available when needed. Legion needs multiple users to be available at the same time in order for its input mediators to work correctly. Prior systems have also needed multiple workers to be available. For instance, the ESP Game encouraged accurate image labels by pairing players together and requiring them both to enter the same label, although ESP Game players could also be paired with simulated players [22]. Seaweed reliably got Mechanical Turk workers to be available at the same time to play economic games by requiring the first worker to arrive to wait (generally for a few seconds) [5]. Legion similarly utilizes the input of multiple workers and asks workers to wait until enough workers have arrived, but engages workers for longer control tasks.

Prior systems have enabled real-time control from the web, most often in the context of robotics [19]. Osentoski *et al.* used a web-based interface to a robot to crowdsource a substantial amount of training data that they then used to train a system for automatic real-time control of a robot [15]. Goldberg *et al.* enabled groups of web users to collectively control a web cam [20] and make navigation decisions for a human actor [9] by interactively choosing regions of interest in captured images. Such systems are generally created only for the control of a particular robot or other system, whereas Legion can be used to control any interface. Legion might help researchers train other types of system for real-time control.

THE CROWD

We define *the crowd* as a dynamic pool of anonymous workers of varying reliability. Because the pool is dynamic, workers come and go, and no specific worker can be relied upon to be available at a given time or to continue working on a job for a set amount of time. Workers cannot be relied upon to provide high-quality work of the type one might expect from a traditional employee for various reasons including misunderstanding of task directives, laziness, or even maliciousness. Finally, workers may experience delays that are beyond their control, such as network bandwidth variability.

For enabling real-time control, the dimensions of the crowd that are most relevant are (i) the time each recruited worker continues working on the task and (ii) the quality of the worker's output. These can be measured empirically for a specific crowd source, but are expected to be task-dependent [21]. A related dimension is the latency required to recruit workers to a particular job. For this paper, we assume that workers can be made available quickly, recruited and kept available using systems like quikTurkit [3].

Our experiments are run on Amazon's Mechanical Turk because of the ease by which workers can be recruited. Nevertheless, our framework is compatible with worker pools from other marketplaces, volunteers drawn from social networks, or any other group of workers available.

CONTROL CONSIDERATIONS

Legion can control a variety of applications, but several dimensions are especially important for enabling real-time crowd control. As we explain later in this paper, Legion synthesizes input from multiple workers by identifying when workers in the crowd provided the same input at the same time. To recognize when multiple inputs agree, Legion requires the input to be discrete, and, to associate input over time, Legion uses fixed time windows. As we will see, this does not mean worker input needs to be delayed until the end of a window.

The input space is defined by the application that is being controlled. GUI applications vary from being controllable by a few discrete keys to using the full continuous input of a pointing device. Key presses are already discrete. Clicks are also discrete in the high-dimensional space of the pixel locations. Legion reduces the size of this space to a fixed grid in order to more easily associated inputs together. Legion does not currently handle other pointer interactions, such as pointer movement paths or dragging.

Many tasks have several correct ways of completing them. For instance, if the task is to navigate a robot around an obstacle to a specified location there are at least two reasonable, high-level paths (going left around the obstacle or going right). Applications can be characterized by the number and degree of these decisions points. Crowd input can be expected to diverge more at such decision points.

To correlate worker inputs over time, we divide time into discrete windows called *epochs* and associate inputs received in the same epoch together. Tasks with more decision points may be more easily solved by mediation strategies that allow for longer term strategies over multiple epochs.

LEGION

Legion is comprised of (i) an end user client application for capturing and controlling interfaces with the crowd, (ii) a server-side framework for recruiting and mediating input from crowd workers, and (iii) a web-based frontend on which crowd workers complete the specified task (Figure 2).

End User Client Application

The Legion client allows users to select a portion of their screen to be controlled by the crowd by drawing a box around it. We chose to allow users to flexibly choose the region of the screen to export, instead of choosing a particular window, because it allows users to (i) exercise control over the information shared with the crowd, (ii) expose simpler interfaces for workers comprising only necessary interface components, and (iii) create new mash-ups by sharing pieces of multiple applications arranged next to one another (discussed later). Smaller regions also lower the required bandwidth.

Users provide a natural language description of the task that they would like the crowd to complete, and create a legend of keys that the crowd can use and where they are able to click. To simulate mouse clicks and keyboard events locally, Legion uses the OS X Application Services framework to post Quartz events into the event stream at specific screen locations. The CamTwist library[1] captures video from the user's screen and sends it to the server. Only specified keys and mouse clicks in the defined region are simulated (we use a white list). This does not completely ensure security, but reduces what workers are able to control on the client GUI. Future work may explore how to further isolate crowd input.

[1] http://www.allocinit.com

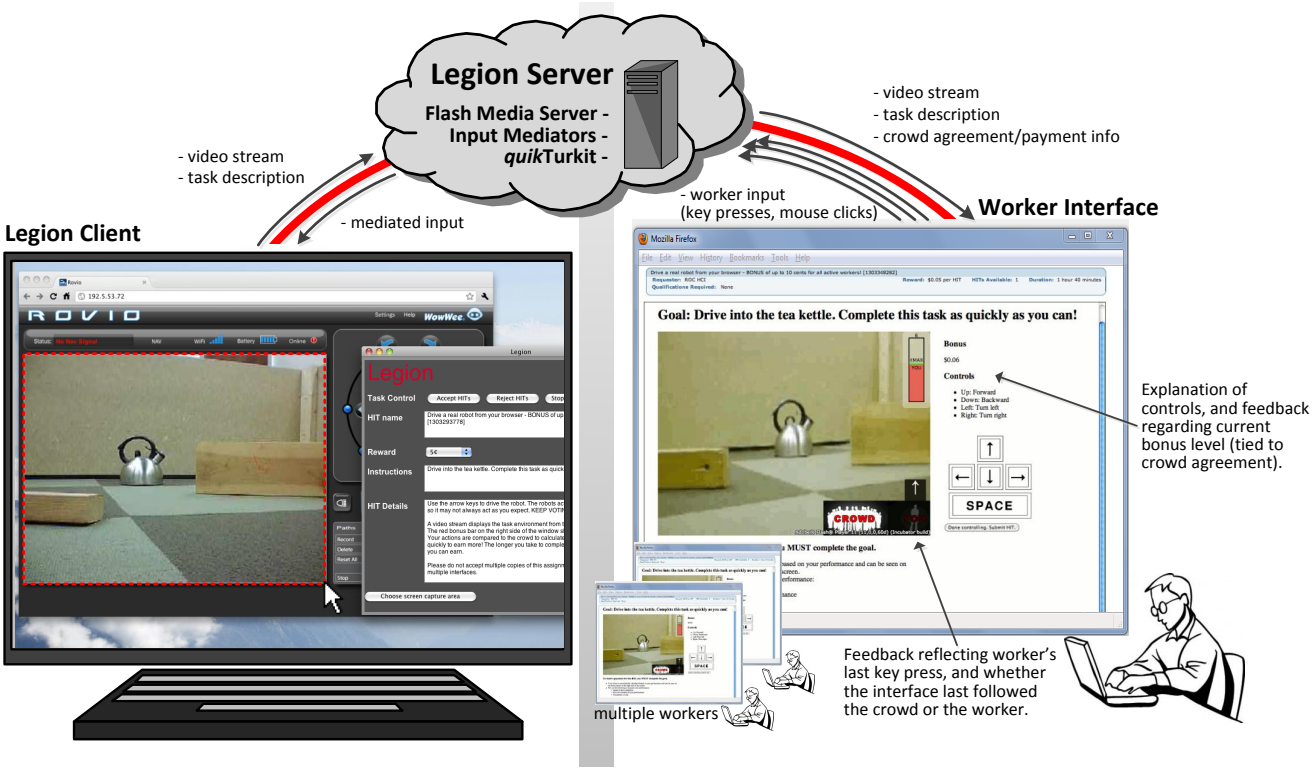

Figure 2: Legion is a framework that allows existing interfaces to be outsourced to the crowd. In this example, a user has outsourced control of her Rovio robot. The Legion client allows end users to choose a portion of their screen to send to crowd workers, sends a video stream of the interface to the server, and simulates events (key presses, mouse clicks) when instructed by the server. The Legion server recruits workers, aggregates the input of multiple crowd workers using flexible input mediators, and forwards the streaming video from the client to the crowd workers. The web interface presents the streaming video, collects worker input (key presses and mouse clicks), and gives workers feedback.

Users decide when the crowd is done, and decide whether the crowd as a whole successfully completed their given task.

Server-Side Framework

The Legion server is a Java application that recruits multiple workers from the crowd; collects, mediates, and forwards worker key presses and mouse clicks to the client application; and pays workers for their work. To recruit workers, Legion uses quikTurkit, a script for the TurKit platform [10], that maintains an active pool of workers. The HITs are described using the short task description provided by the end user. Workers can either be recruited on-demand or automatically when the user opens the application in anticipation of future need. If workers arrive early they are paid to wait. quikTurkit was able to satisfactorily maintain the dynamic pool of at least 3-5 workers needed for Legion.

The Legion server also includes Flash Media Server (FMS)[2]. Video is streamed from the client application to FMS, which supports multiple workers receiving the stream via their web browsers. The video is compressed using On2.VP6 and sent using the User Datagram Protocol (UDP), which unlike TCP, allows packets to be dropped. As a result, if bandwidth is temporarily reduced between our server and the workers, frames will drop instead of being queued, helping to ensure that the frame currently being shown is the most recent one.

[2]http://www.adobe.com/products/flashmediaserver/

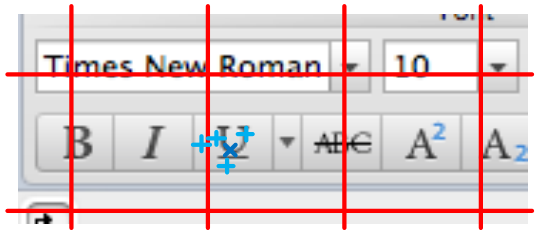

Figure 3: Our method for discretizing mouse clicks. The screen is split into a grid without knowledge of the interface beneath it, and so some inputs that are semantically the same will be discretized into different grid cells (as here). When simulating the click, Legion either uses the coordinate directly or averages the positions of each, depending on the input mediator.

Worker key presses and mouse clicks are sent to the server, which aggregates them and uses one of several input mediators to choose which events to send to the client. The server framework allows inputs to be analyzed, aggregated and filtered in a number of ways. It includes a number of procedures for deciding which inputs to pass through, for blocking or allowing inputs only from certain workers, or for analyzing inputs over time windows. End users will neither write these procedures nor need to decide between them, but the flexibility of the framework may provide opportunities for researchers looking to evaluate different strategies for mediating input from multiple crowd workers using closed-loop control.

The input mediators require input to be discrete. This is straightforward for key presses. To discretize mouse clicks, we divide the screen into a grid and use the grid cell in which the click occurred (Figure 3). For instance, the event descriptor *mc_12_20* refers to a mouse click in the grid cell at (12,20). Later, this can be generalized by clustering the mouse clicks in order to find a discrete set of possible actions to select from. Discrete inputs allows the input mediators to compare the inputs of crowd workers.

The input mediators that we have implemented are described in the next section.

Worker Web Page

Workers control the interfaces via a web page hosted by the Legion server. This web page displays the live video feed of the client interface, and collects the key presses that workers make and sends them back to the client. As workers are queued, they play a simple game in which they are shown a letter and asked to type what it says. Although we don't currently, we could use this as a simple test to weed out workers who provide bad input or whose latency is too high.

Providing good feedback is difficult because a worker's input will often not be followed. In the robot control case, for instance, a worker may tell the robot to turn right, but the robot may go left because that action is preferred by other workers. As users press keys or click the mouse, their input is reflected back to them visually in the interface (Figure 2). They also see whether their input or the crowd's input was last sent to the client. We explored showing the workers the chosen input (e.g. the *m* key), but workers quickly learned to mimic it to improve their crowd agreement score. Workers are shown their current crowd agreement score in a power meter, a fractional value serving as a multiplier against the maximum bonus they can receive (in our case 10 cents). Workers are not told directly whether they are the current leader (in input mediators that support it), but they may be able to infer their status.

CROWD CONTROL

Legion aggregates the control input from all of the workers in the crowd into a single input that is forwarded to the interface as if it was sent by a single user. We developed the five input mediators described below. Each has strengths and weaknesses, which we expect to be application dependent and compare later in the paper. The input mediators each balance reliability and latency differently. For example, the *solo* input mediator recruits a single worker to directly control the interface. Latency will be low but so will reliability.

Mob The mob input mediator simply serializes the inputs from each of the workers and sends the single stream of actions to the interface. Each input from a worker is immediately sent to the interface being controlled. This approach may work for systems in which large numbers of redundant inputs are either ignored or handled gracefully. In this case, the 'wisdom' of the crowd is preserved by the fact that the majority of the inputs to the system will be those most agreed upon by the crowd. For applications in which excess or redundant input leads to loss of accuracy, such as editing a document, this style of mediation will perform poorly.

Vote The vote input mediator attempts to address the problem of unreliable individual crowd workers. We use a weighted vote, in which each user has a corresponding weight that acts as an influence measure. At the end of each epoch worker's inputs are collected as votes and scaled based on the weight of the worker who cast the vote, and summed to find the action with the highest weighted value amongst all participating workers. This action is then sent to the client and the weights of each worker w_i are recomputed according to the following formula:

$$w_i^{(t+1)} = \alpha w_i^{(t)} + (1-\alpha)\frac{\sum_{j=1}^{N_{A_i}} w_j^{(t)}}{N} \quad (1)$$

Where t is the current epoch, N_{A_i} is the number of workers that voted for selected action A_i and N is the total number of workers casting votes. α is a discount factor selected such that $\alpha < 1$. Its effect is that a worker's influence is affected more by recent agreement with the crowd than historical agreement.

We expect that using worker inputs as votes will improve accuracy, but at the cost of slowing down response time. This is because the votes of crowd workers will not be temporally synchronized, meaning that epoch windows can only be reduced so far before corresponding votes start falling into different epochs, thus skewing the vote. We used an epoch of 1 second in our experiments.

Leader In order to reduce the latency inherent in gathering votes over the span of an epoch, the *leader* input mediator selects the highest influence worker at the beginning of each epoch to assume direct control for its duration. This means that each input entered by the leader is immediately forwarded to the interface being controlled. Since the leader is elected based on weight, they serve as leader for as long as they remain in agreement with the crowd, on average.

The leadership model provides a means for real-time responses to feedback from the system without sacrificing the benefits of crowd agreement and enables longer term plans. For example, suppose a navigation task requires a decision to be made as to which path to take in order to avoid an obstacle. In the *vote* or *mob* input mediators, close crowd decisions can result in actions belonging to disparate plans being performed in a sequence because of crowd weight and participation fluctuations. This may lead to a course of action which is not in alignment with any individual worker's plan. In the navigation example, this may result in colliding with the obstacle. Electing a single leader allows them to take consecutive actions coinciding with their individual plan.

Worker weights are calculated using a *bag-of-votes* model to choose a leader after each epoch. These weights are calculated by comparing the normalized actions of each worker to the actions of the crowd with the vector-cosine as follows:

$$VC(a_i, c) = \frac{a_i \cdot c}{||a_i|| * ||c||} \quad (2)$$

where a_i is the i^{th} worker's normalized action set and c is the crowd's. We then recompute the worker's weight after each epoch similar to before, but use the vector-cosine as the new

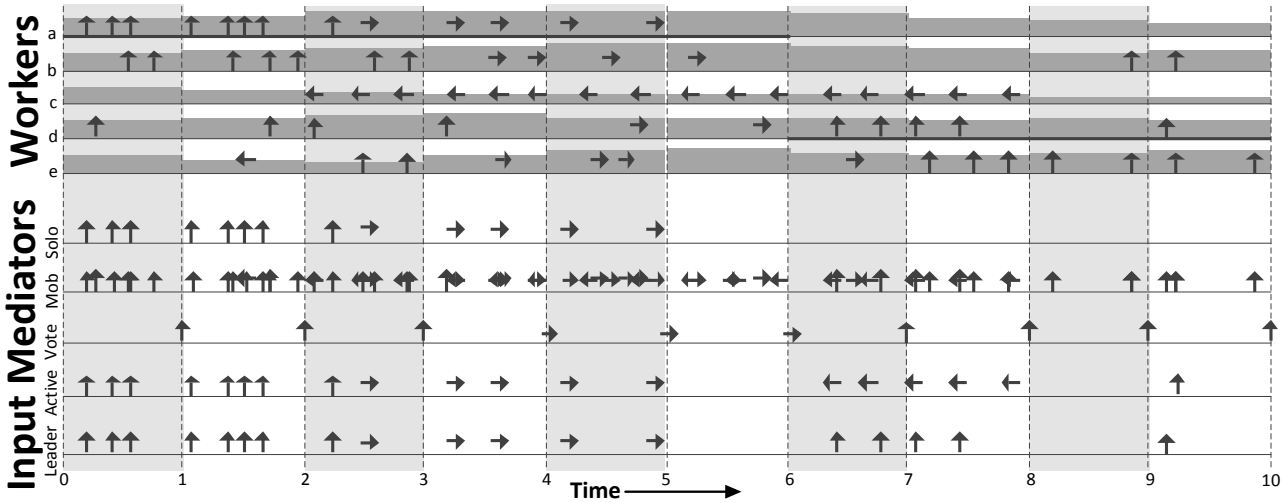

Figure 4: A visual depiction of the five input mediators. The application is controlled using only the left, right, and up arrow keys. We assume the "best" strategy to be *up* for epochs 0-3, *right* for epochs 4-5, and then *up* again from 6-9. Workers *a, b, d, e* all generally exercise this strategy, although none vote during every epoch. Worker c misunderstands the task, is malicious, or is trying to solve the task in a different way by pressing only *left*. The shading indicates the worker's crowd agreement level for the given epoch. *solo* chooses worker a at random and accepts input only from that worker, which works well until that worker stops providing input. *mob* serializes and passes through all input, without considering crowd agreement. *vote* issues the single most popular input over an epoch at the end of the epoch. *active* works like *solo* but switches to a new worker when the chosen worker does not vote during an epoch. *leader* dynamically chooses the worker with the highest crowd agreement level at the beginning of each epoch and passes their input through immediately.

crowd agreement metric:

$$w_i^{(t+1)} = \alpha w_i^{(t)} + (1-\alpha) VC(a_i^{(t)}, c) \quad (3)$$

where $\alpha < 1$ is the same discount factor as in Eq. 1.

Active The *active* input mediator is variation on *leader* that we created in order to tease apart the two benefits of *leader*: (i) control by a single worker in the crowd, and (ii) the selection of a worker that has been shown to agree the most with the crowd. The *active* input mediator randomly selects a worker, who maintains control as long as they continue to provide input. If they fail to provide input during some number of consecutive epochs (here we used 5), a new random worker is selected.

Mediation Strategies Not Explored There are clearly a number of mediation strategies beyond the five described here. In particular, we did not explore hierarchical input mediators in which different workers have different roles. For instance, some workers could be responsible for controlling the interface and a separate group of workers could vote whether (or even which of) the workers were doing a good job. Our input mediators also do not allow workers to indicate their confidence in the actions they suggest, for instance determining that they are so confident of an action that they would like to wager that it will end up being a good action to take in the long run. We could imagine variations on the strategies above that allow limited communication between workers to help them devise and exercise long-term plans.

MEDIATOR EVALUATION

We evaluated our mediation strategies on two applications: robot navigation and data entry into a spreadsheet. These applications varied in characteristics that we hypothesize will manifest in the mediation strategies. Specifically, the robot task has a small input space and is designed such that workers can follow only one general path to complete it. The spreadsheet entry task has a large input space (all keys and mouse clicks) and can reasonably be completed in different orders.

We paid workers 5 cents per task in our experiments if the crowd completed the task. Workers could earn up to 10 more cents based on their crowd agreement level and the total time taken to complete the task. We waited for at least 3 workers to be available before starting the task, which generally took less than one minute.

As an initial feasability test, we used Legion to control the web application in Figure 5 to measure latency. The application displayed a random key, crowd workers typed it, and then repeated. They were not shown the latency measurements. Timing is done in the web application, which we ran on our machine, and sums both system and human latency on this simple task. On average the recorded latency was 854.6 milliseconds (SD=743.0). This gives a sense of the lowest overall latency supported by Legion on Mechanical Turk, although realized latency may be task dependent.

Robot Navigation

Robot control is natural as a motivating task because it exercises three main contributions of Legion: robots need to be controlled in real-time, creating a new robot control interface for web workers would require significant one-off programming effort, and no obvious way previously existed for multiple crowd workers to simultaneously, collectively control robots in real-time. Robot control is difficult, so enabling even simple navigation tasks can require substantial customization for the robot and the environment. Even rela-

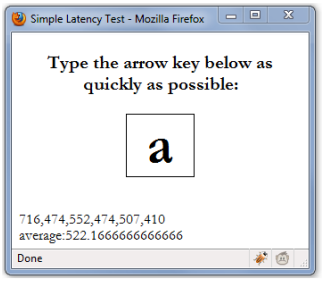

Figure 5: Web application used to measure latency.

tively simple tasks can become complex - the robot we used drifts severely to the left when told to go straight. Completing tasks with a robot can require longer-term strategies to be executed, e.g. it may need to go around a barrier, moving farther from its goal before reaching it. Finally, tasks naturally take some time to complete, and so workers available at the start might leave before the end. Eventually, our aim is to use crowds to control assistive robots that both navigate through and manipulate the environment.

We used an inexpensive remote-controlled mobile webcam called Rovio[3] as a platform for experiments with robot navigation. This device can be controlled over wifi via its web-based interface (Figure 2). Although it is marketed as a "robot," it does not contain any functionality for automatic decision-making or navigation. By connecting the Rovio to the crowd with Legion, we created an intelligent mobile robot that accepts natural language commands.

The navigation task was to drive the robot from a start position into a tea kettle a few feet away. Although the goal could be seen from the start of the task, the direct path to it was obstructed by various objects (Figure 2). Legion captured video at 320x240 resolution. As a baseline, we asked three local students to complete the robot navigation task three times as quickly as possible. On average these local workers required 46.3 seconds (SD=12.4) to complete the task.

We ran 10 trials for each of the five mediation strategies. The total unique voters (those who voted at least once) varied from 1 to 14 per task, although on average 3.1 workers voted during each epoch. These numbers highlight the highly dynamic nature of the crowd on Mechanical Turk. There were not detectably significant differences in the number of engaged workers across conditions. We ended trials that lasted more than 10 minutes.

In both the *active* and *leader* conditions, all trials were successfully completed, as compared to 8/10 successful trials with *vote* and *mob*, and only 4/10 successful trials with *solo* (Figure 7). Few trials were completed successfully with *solo* because workers who did not quickly complete the task disconnected. The crowd seemed unable to exercise a consistent strategy with the *vote* and *mob* input mediators. *vote* was further hampered by its slow rate of issuing actions.

We also considered task completion time (Figure 6). When the chosen worker completed the task, *solo* was the fastest, averaging just 56.0 seconds (SD=12.9). Trials in the *leader*

[3]http://www.wowwee.com/en/support/rovio

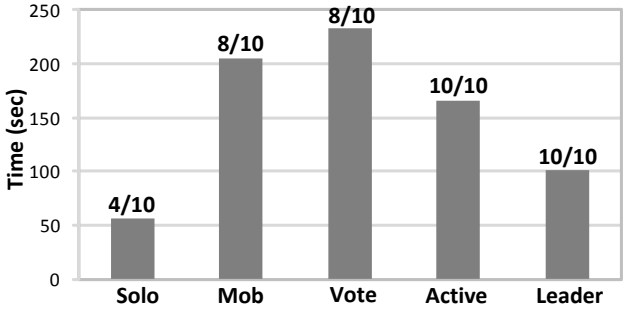

Figure 6: The average time required to complete the robot navigation task for each of the input mediators, along with the fraction of successful runs. *solo* was the fastest but only completed 4 of 10 runs, whereas *leader* was close in time and completed all runs. Choosing leaders based in crowd agreement in *leader* outperformed *active*, which chooses leaders randomly.

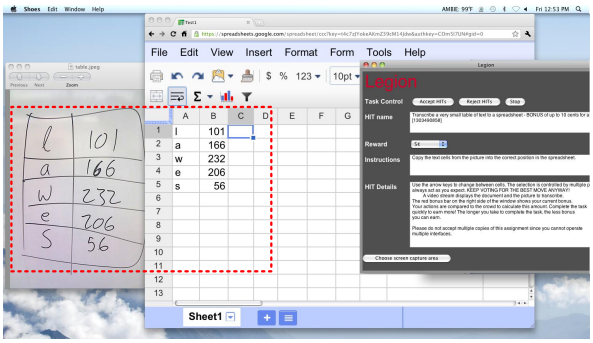

Figure 8: User view of spreadsheet transcription task.

condition were completed faster than trials in the *active* condition, 101.7 seconds (SD=81.0) vs 165.7 seconds (SD=166.3), a significant difference ($F_{1,9}$=6.96, $p < .05$). This suggests that choosing the leader based on crowd agreement, rather than randomly, leads to better results. *vote* performed no better than *mob*, 232.4 seconds (SD=110.5) vs. 205.8 seconds (SD=140.1), a difference that was not detectably significant.

Spreadsheet Transcription

Our next experiment explored the *vote*, *active*, and *leader* mediators on a simple spreadsheet transcription task (Figure 8). Prior work has demonstrated the value and feasibility of crowd assistance in word processing tasks [2]. In this experiment, we used Legion to capture both a Google spreadsheet and a picture of a table scribbled on a whiteboard (the timing results from the robot navigation task), and workers were asked to transcribe the table into the spreadsheet. Legion captured video at 640x480 resolution. This task is interesting because of the large input space (numbers, letters, arrow keys, and mouse clicks), which makes it more difficult for the crowd to agree. Furthermore, while Google spreadsheets already enable collaborative use, there is no protection against malicious users. As such, it is not suitable for collaborative use by the crowd.

We again conducted 10 trials of each of the input mediators, and ended trials lasting more than 10 minutes. Trials were deemed successful when all of the labels and numbers were entered into the spreadsheet correctly.

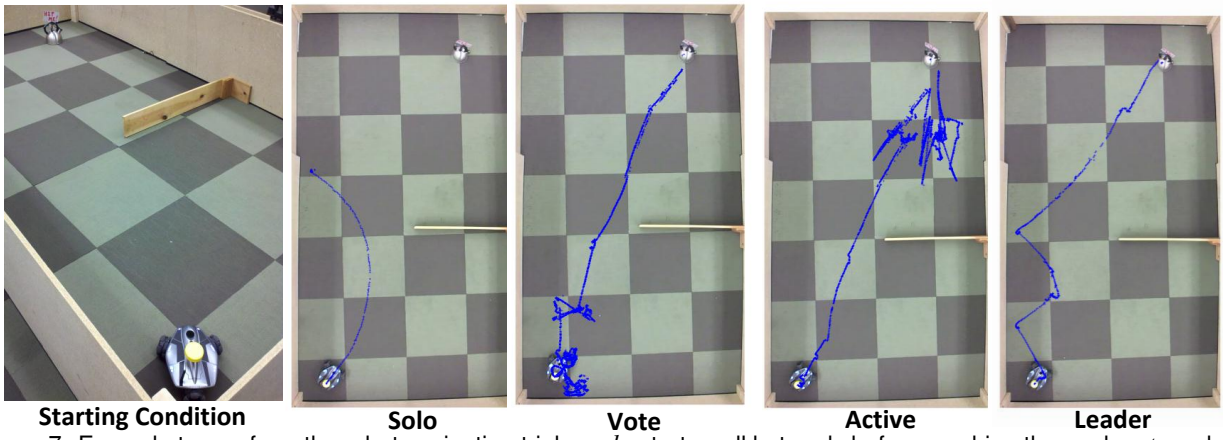

Figure 7: Example traces from the robot navigation trials. *solo* starts well but ends before reaching the goal, *vote* makes shaky progress slowly toward the goal, *active* goes toward the goal but fails to reach it when the worker in control drives back and forth near it, and *leader* goes mostly straight toward the goal (the robot tends to go left).

None of the *vote* trials were successful, whereas 9 of 10 trials with both *active* and *leader* were successful. With the *vote* input mediator, it was particularly difficult for the crowd to converge on a single plan. We did not test the *mob* mediator because we expected similar performance, and we expect the *solo* mediator to again perform like *active* but with fewer successful completions. Task completion times were again lower for the *leader* condition as compared to *active*, 78.2 seconds (SD=72.3) vs. 100.0 seconds (SD=74.8), although this difference was not detectably significant.

VALIDATION IN NEW TYPES OF APPLICATIONS

Legion enables end users to flexibly and creatively apply crowdsourcing to their existing interfaces. In this section, we explore several new types of applications enabled by Legion that we created using the framework. Although additional work is necessary to fully evaluate the utility of these applications, we present them here as a demonstration of the broad set of possibilities Legion opens up.

Interactive End User Assistance

Most of our discussion of Legion has assumed that a user will outsource full control of her interface to the crowd, but Legion can be used much more flexibly to provide interactive assistance to end users.

User Support Legion enables end users to outsource certain functions to the crowd who then can work cooperatively with them as they work. We used this idea to help make a crowd-powered predictive keyboard out of an existing predictive keyboard software program called KeyStrokes[4]. On-screen keyboards are used by many people with motor impairments, who use trackballs, head pointers, and other devices to type. Because typing in this way can be slow, KeyStrokes predicts what word the user is likely trying to type and displays suggestions that can be selected with a mouse click. We used Legion to outsource the selection of suggestions in order to bring human intelligence to prediction and possibly make typing faster (Figure 9). When using this application, we found workers would often disrupt our typing by choosing suggestions prematurely; applications of this type clearly need to manage the initiative between users and the crowd.

[4]http://www.assistiveware.com

Co-Pilot Mode In co-pilot mode, the end user controls the interface herself with the crowd standing by in case she needs to temporarily attend to something else. We implemented this in the Legion framework by creating a modified version of the Legion client that captures and sends local user input to the server, a modified version of the server that can accept input from the local user in addition to the crowd, and a modified *Leader* input mediator that artificially gives the end user a weight high enough that she will always be the leader.

The result is that a user can control her interface as she normally would, but the system will automatically and quickly transition control to the crowd when she leaves. This mode may be particularly useful in real-time gaming or other applications that require continuous attention and control. Currently, users must stay at their computers, or signal they will be away from keyboard (afk).

The Co-Pilot Apprentice An interesting extension of co-pilot mode is for the end user to train the crowd to control an interface. Because the end user is always the leader through an artificially high weight, the best way for the crowd to increase their agreement score (and receive a higher bonus) is to mimic what the end user does. The co-pilot application can thus be used to program the crowd by demonstration.

Programming by Demonstration

We used Legion to enable basic programming-by-demonstration across diverse applications. The eventual goal is for automatic systems to learn to complete new tasks with existing interfaces by watching the crowd complete those tasks, but as a proof of concept we implemented a simple recording mechanism on the Legion server that can capture the inputs provided by the crowd and then replay them. We successfully recorded and replayed tasks with both the Rovio robot and Excel.

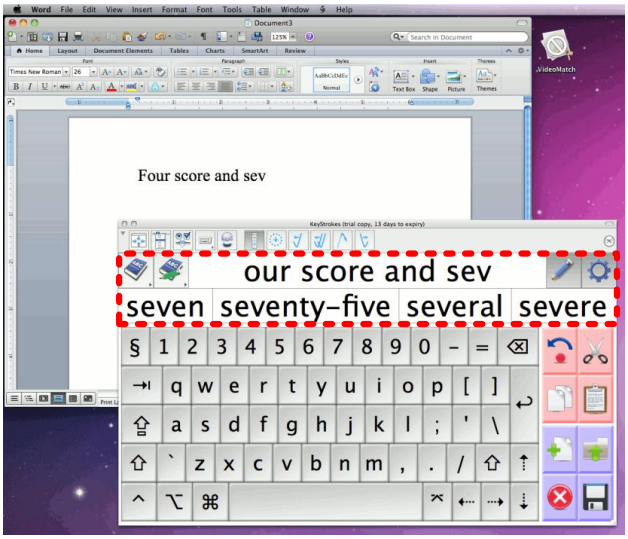

Figure 9: In this Legion use case, suggestion selection in a software keyboard is crowdsourced while the user continues to control the software keyboard.

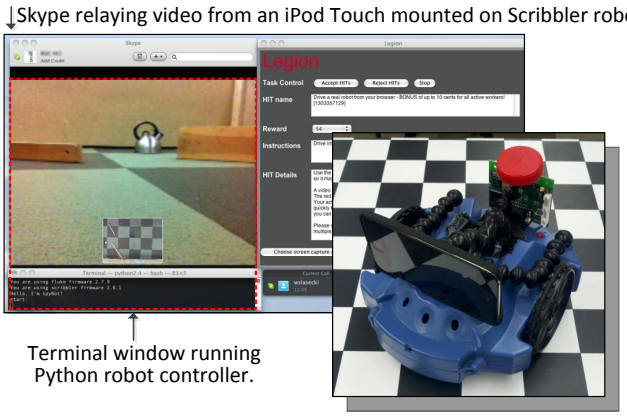

Figure 10: A crowd-powered robot created by mounting an iPod Touch to a Scribbler robot. Legion crowdsources the Skype video and the controller terminal.

Mash-Ups of Desktop Applications:
Finally, we explored how Legion could be used to create a novel type of desktop mash-up in which pieces of multiple existing interfaces are sent to the crowd for control.

We have already seen an example of this in the second experiment in the previous section, in which we combined a simple photo viewer and Excel to enable the crowd to fill in a spreadsheet with the numbers sketched on a whiteboard.

As a second example, we created a new video-enabled robot by combining a remote-controlled Scribbler[5] robot and an iPod Touch running Skype (Figure 10)[6]. The Scribbler driving platform was controlled over bluetooth from a terminal window. To construct the robot, the iPod was simply taped to the platform. On the desktop, we moved the Skype and terminal windows close together and then used the Legion end user interface to select the relevant parts of both of these windows. This mash-up allowed workers to see where the robot was going, and type commands to control it.

[5]http://www.parallax.com/tabid/455/Default.aspx
[6]This robot actually performed better than the Rovio.

DISCUSSION

We have introduced Legion, a system for real-time crowd control of existing interfaces. Although Legion can control a wide variety of interfaces, our experiments highlighted the fact that different input mediators may be appropriate for different types of applications. Applications with a large input space, such as the spreadsheet, proved most difficult for the input mediators that did not select a single individual. Tasks in which the crowd was presented with multiple reasonable courses of action and a large input space made it especially difficult to achieve high crowd agreement levels.

The input mediator that did consistently well was the *leader* input mediator, which elects a leader who has direct control over the interface as long as he remains in power. This would go against the notion of the *wisdom of the crowds*, if the leader had not been elected by the crowd. Nevertheless, input mediators that allow a single crowd worker to control an existing interface trade expediency for trust. As a result, applications in domains in which errors have consequences may need to trade higher latencies for reliability.

It was clear that the crowd faced challenges related to missing feedback. Because multiple workers were controlling the same interface, a worker's actions would not always be reflected in the behavior of the interface. We received several emails from workers wanting to let us know that they had been trying to control the application as instructed but that it did not seem to be following their instructions. These concerns may dissipate as workers become accustomed to interacting in this way, but we may also need to find ways to give better feedback via the interface. Such concerns may have real implications, as we suspect that workers who felt that they were not being listened to quit the task earlier. Our current interface shows workers when the crowd decision was taken over their own, but it would be nice to give users the impression that the interface followed their input.

Legion does not support communication between workers, which is unusual for collaborative systems. Early experiments that showed workers what action the crowd chose resulted in poor quality input as workers mimicked what the crowd was already doing. Nevertheless, watching the crowd seem to struggle against one another to complete tasks suggests that a form of limited communication may be helpful.

Our experiments show that it is possible to have multiple crowd workers collaboratively control an existing interface in real-time. Our experiments were conducted on Amazon's Mechanical Turk, and many of our workers came from countries other than the United States. We expect that other crowds would face fewer technological limitations.

FUTURE WORK

Future work may explore how to better enable group work in Legion. For instance, new input mediators may help facilitate consistency. Our *leader* input mediator enables a consistent strategy as long as the same leader remains in control. Intelligent input mediators might be able to automati-

cally cluster input into distinct strategies, allowing leaders to be replaced by other workers likely to execute similar plans. Other promising approaches include allowing workers limited communication, or to enforce a management hierarchy.

We are currently working on ways of extending Legion to more robustly and automatically adjust to different domains. It may be useful, for example, to dynamically switch between multiple input mediators. For instance, the *mob* could normally provide control with the benefit collective crowd voice, but *leader* could take over at a decision point at which the collective voice might not agree.

Complex applications may be better controlled by multiple specialized groups (Figure 1). For instance, an assistive robot requires control for both navigation and manipulation; each function may be best handled by different sets of crowd workers. For some tasks, limited communication between workers may facilitate collaboration. Finding effective ways of moderating communication to avoid excessive or malicious messaging in this context is ongoing.

As explored briefly in this paper, Legion can be used as a basis for interactive programming by demonstration. Future work may look to better support end users training a crowd to help them on their tasks, or new input mediators that will continue to reward the crowd for exercising control similar to what the end user demonstrated even after the user has left.

Workers often found the feedback provided by Legion confusing. Because the control input sent to the interface is based on the crowd, an individual may find that the interface does not do what they tell it. It may be interesting to explore how to allow different users to take different paths. In some systems with real-world consequences this would not be possible, e.g. in the robot domain, but for many others it may be possible to simulate the effects of different actions being sent to the interface. For instance, copies of the application could be run simultaneously in virtual machines and only merged when the crowd's consensus was clear. We also plan to explore giving workers more information about their current status, for instance whether they are the current leader.

CONCLUSION

We have presented Legion, a system that enables real-time control of existing interfaces by the crowd. Prior approaches to crowdsourcing require programmers to encapsulate tasks into new interfaces, cannot be used for continuous control, and use abstractions for reliability that introduce additional latency. We have implemented and evaluated several ways of combining input from multiple workers in real-time, and have demonstrated how letting the crowd control existing interfaces allows for several new kinds of applications.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Craig Harman and Robin Miller.

REFERENCES

1. J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom. Plow: a collaborative task learning agent. In *Proc. of the national Conf. on Artificial intelligence - Volume 2*, 1514–1519. AAAI Press, 2007.
2. M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich. Soylent: a word processor with a crowd inside. In *Proc. of the ACM Symp. on User interface software and technology*, UIST '10, 313–322, 2010.
3. J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh. Vizwiz: nearly real-time answers to visual questions. In *Proc. of the annual ACM Symp. on User interface software and technology*, UIST '10, 333–342, 2010.
4. P. Chan, and S. Stolfo. Learning Arbiter and Combiner Trees from Partitioned Data. In *KDD 1995*, 39–44, 1995.
5. L. Chilton. Seaweed: A web application for designing economic games. Master's thesis, MIT, 2009.
6. S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popovic, and F. Players. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.
7. M. Dixon and J. Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proc. of the Intl. Conf. on Human factors in computing systems*, CHI '10, 1525–1534, 2010.
8. S. Greenberg and D. Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proc. of the ACM Conf. on Computer supported cooperative work*, CSCW '94, 207–217, 1994.
9. K. Goldberg, D. Song and A. Levandowski. Collaborative teleoperation using networked spatial dynamic voting. In *Proceedings of the IEEE*. 91(3): 403-439, 2003.
10. G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: human computation algorithms on mechanical turk. In *Proc. of the ACM Sypm. on User interface software and technology*, UIST '10, 57–66, 2010.
11. G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *Proc. of the Conf. on Human factors in computing systems*, CHI '07, 943–946, 2007.
12. D. Maynes-Aminzade, R. Pausch, and S. Seitz. Techniques for interactive audience participation. In *Proc. of the Intl. Conf. on Multimodal Interfaces*, ICMI '02, 15–, 2002.
13. Massively multiplayer pong. collisiondetection.net, 2006.
14. J. Nichols, Z. Hua, and J. Barton. Highlight: a system for creating and deploying mobile web applications. In *Proc. of the ACM Symp. on User interface software and technology*, UIST '08, 249–258, 2008.
15. S. Osentoski, C. Crick, G. Jay, and O. C. Jenkins. Crowdsourcing for closed loop control. In *Proc. of the NIPS Workshop on Computational Social Science and the Wisdom of Crowds*, NIPS 2010.
16. D. Opitz, and R. Maclin. Popular Ensemble Methods: An Emperical Study. In the *Journal of Artificial Intelligence Research*, 11: 169–198, 1999.
17. T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2:33–38, Jan/Feb 1998.
18. R. W. Scheifler and J. Gettys. The X Window System. *ACM Trans. Graph.*, 5:79–109. April 1986.
19. D. Schulz, W. Burgard, A. B. Cremers, D. Fox, and S. Thrun. Web interfaces for mobile robots in public places, 2000.
20. D. Song, A. Pashkevich and K. Goldberg. ShareCam Part II: Approximate and Distributed Algorithms for a Collaboratively Controlled Webcam. In *Proc. of the IEEE Conf. on Intellignet Robots and Systems*, IROS '03, 1087-1093, 2003.
21. M. Toomim, T. Kriplean, C. Prtner, and J. A. Landay. Utility of human-computer interactions: Toward a science of preference measurement. In *Proc. of the ACM Conf. on Human Factors in Computing Systems (CHI 2011)*, CHI '11, 2011. To Appear.
22. L. von Ahn and L. Dabbish. Labeling images with a computer game. In *Proc. of the SIGCHI Conf. on Human factors in computing systems*, CHI '04, 319–326, 2004.
23. T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using gui screenshots for search and automation. In *Proc. of the ACM Symp. on User interface software and technology*, UIST '09, 183–192, 2009.