Cerebro: Programming Opportunistic Interactions Across People

Jennie Werner, Ryan Louie, Allison Sun, Kapil Garg, Ryan Madden, Kevin Chen, Shannon Nachreiner, and Haoqi Zhang
Northwestern University
Evanston, IL
{jenniewerner2018, ryanlouie, allisonsun2018, kapilgarg2017, ryanmadden2017, kevinchen2016, shannonnachreiner2012}@u.northwestern.edu, hq@northwestern.edu

ABSTRACT
We explore the design of *opportunistic collective experiences* (OCEs), programs that facilitate opportunistic interactions between people across distance and time. This paper introduces Cerebro, a computational platform for expressing and facilitating interactions through OCEs. Cerebro consists of: Affinder, a programming environment for expressing rich situations; the Collective Experience API for programming opportunistic interaction structures; and the Opportunistic Interaction Engine, which runs OCEs by connecting participants as situations arise. Through two exploratory studies, we demonstrate how Affinder allows OCE designers to express complex situations, and how OCEs promote shared engagement between users.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous;

Author Keywords
opportunistic interaction; opportunistic collective experiences; context-aware computing; social and crowd computing

INTRODUCTION
We increasingly rely on social technologies to help us connect with family, friends, and strangers when we are not in the same place or available at the same time. Despite having a plethora of social technologies available for connecting at a distance, meeting both our desire for direct engagement and our need to fit these interactions into our busy lives is still challenging. On one end of the spectrum, social media technologies significantly reduce the cost of sharing at a distance by making it easy to share the contents of our daily lives. While such technologies provide us with constant awareness of what's happening in one another's lives, they do little to support our direct engagement with one another through shared conversations and activities. On the other end of the spectrum, computer-mediated technologies promote direct engagement by allowing for live, rich media conversations at a distance and providing support for coordinating in-person and virtual meetups. But with our busy lives, it is increasingly difficult to find time for serendipitous meetings with one another, and the planning cost of making up for this deficit remains high.

To fill this gap, we explore the design of social technologies for promoting *opportunistic interactions*, or serendipitous moments of shared engagement that can occur without planning. In person, we might bump into an old friend at the bar and share a drink, or have a chat with a co-worker we run into by the water cooler. We consider what it might mean to support such opportunistic interactions across distance, so that people

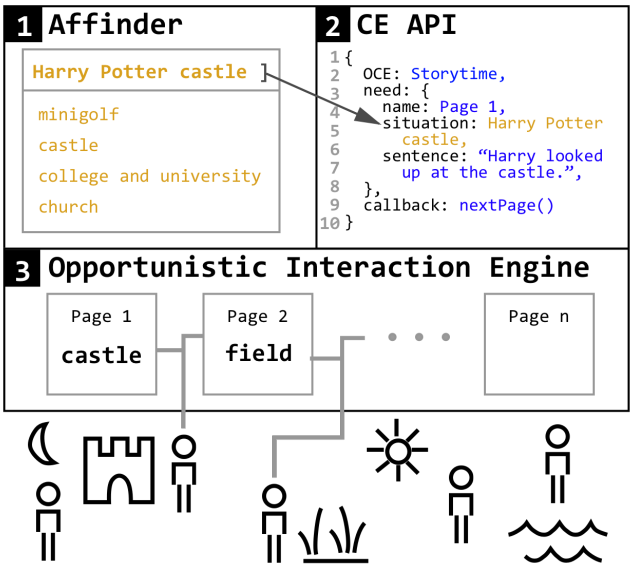

Figure 1. Cerebro is a computational platform for expressing and facilitating opportunistic interactions between people through *opportunistic collective experiences* (OCEs). OCEs are written by using *Affinder* to express rich situations that can be detected automatically across geographically distributed context (1), and the *CE API* to program the interaction and experience progression structure (2). The *Opportunistic Interaction Engine* runs OCEs by monitoring participants, coordinating contributions at opportune moments, and progressing experiences (3).

UIST'18, October 14–18, 2018, Berlin, Germany

ISBN 123-4567-24-567/08/06…$15.00

DOI: http://dx.doi.org/10.475/123_4

who are geographically or temporally distant may nevertheless, at opportune moments in their daily lives, engage with one another in shared activities and experiences. This motivates the need for social technologies that make visible the many serendipitous opportunities for interacting at a distance that currently go unnoticed, and that provide programmatic support for facilitating connecting across these opportunities. For instance, such technologies may support friends 'bumping' into one another when they are in a similar situation or having a similar experience (e.g., waiting in line; or taking their pet to the vet). They may also support new forms of opportunistic interactions, that perhaps take advantage of our distributed context in support of a shared activity. For example, we might envision support for a serendipitous ramen slurping contest with strangers all over the world, or building a storybook with friends who opportunistically contribute to the story based on situations they encounter in their day.

This paper thus introduces *Cerebro*, a computational platform for expressing and facilitating opportunistic interactions between people across time and distance; see Figure 1. Cerebro provides a programming environment that support the rich expression of situations for engaging in opportunistic interactions, and the concise description of the structure of a collective experience that link (multiple) opportunistic interactions across individuals. We call the resulting programs *opportunistic collective experiences*, or *OCEs*. Cerebro then provides automated supports for running OCEs, by recognizing opportunities in which individuals can participate in these OCEs, facilitating participation through a front-end user interface, and progressing experiences as they occur. We hypothesize that such computational supports can significantly expand the number and kinds of opportunities for shared engagement, and generally provide new models for interacting at a distance.

Cerebro consists of three core components for writing and running OCEs:

- *Affinder*, a block-based programming environment that supports expressing conceptually rich situations that can be detected automatically across geographically distributed contexts.

- *Collective Experience API*, or *CE API*, which enables creators to program the interaction structure for opportunistic interactions across situations and describe how these interactions progress as people participate.

- *Opportunistic Interaction Engine*, which runs an OCE created using Affinder and the CE API. The engine automatically notifies users of opportunities to participate and manages the overall experience as an OCE progresses.

Our core conceptual contribution is the idea of *providing expressive languages and automated supports for realizing opportunistic interactions across convenient moments in our daily lives.* Based on this idea, this paper contributes the design and implementation of Cerebro, an end-to-end system for expressing, running, and interacting with OCEs. Cerebro is powered by our core technical contribution, which introduces a *new programming model and associated run-time engine for opportunistic interactions.* Unlike prior crowd programming models that program the assignment of tasks to people, our model supports programming with people's situation directly; this allows the programmer to describe the situations that are conducive for participation, and for opportunities to interact with the system to emerge naturally at run-time. We extend these contributions with two exploratory studies ($N = 7$, $N = 32$), that respectively investigate how experience creators can use Cerebro to express conceptually rich situations, and how end-users interact with one another through OCEs.

In the rest of this paper, we review related work from context-aware computing for representing rich situations and from social and crowd computing on programming with people. We then provide an overview of OCEs, and introduce the components of Cerebro for writing and running OCEs. We present the results of our two user studies with experience creators and end-users, and conclude with a discussion of our research vision for *computer-facilitated opportunistic interactions*.

RELATED WORK

We position Cerebro with respect to work in related two areas: (1) context-aware computing; and (2) social and crowd computing.

Context Aware Computing

Connecting individuals through OCEs require support for expressing rich situations for opportunistic interactions that can be detected automatically by machines across geographically distributed contexts. Our system builds upon years of research and development in context aware computing, that have significantly expanded the ability of computing systems to understand rich human context across the dimensions of location, identity, activity, and time [1]. But unlike prior work that have focused primarily on advancing individual component detectors for various facets of context through better algorithms and sensors (e.g., [13]), we take a *compositional* approach that seeks to advance human supports for composing such detectors to express high level concepts about situations (e.g., "people are enjoying an unusually warm day during winter by exercising outdoors"). In doing so, we leverage in our system a large set of *context features* that are made widely available through public APIs that implement a wide range of component detectors.

While prior work on context-aware programming [6, 5] and trigger-action programming [22] provide tools for users to write triggers and rules based on context features, these tools effectively provide access to the context features directly, and are not designed to support expressing conceptually-rich situations that may be several levels of abstraction removed. For example, while multi-trigger variants of IFTTT (if-this-then-that) support parameterizing multiple context features into a single event, the situations expressed are assumed to be near the sensor level, which allows rules to be built by accessing the conceptual features directly [22]. Further, available end-user programming tools are generally designed with a single end-user in mind, and do not provide supports for describing general situations that can trigger in a variety of geographically distributed contexts across individuals.

To fill this gap, our work studies the design of effective processes and tools for *constructing* rich, opportunistic situations with context features. In one direction, a top-down process can support a creator decomposing situations into simpler concepts. But in early prototypes of Affinder, we found that this can frequently lead to scenarios where a creator finds that there are no matching context features for such concepts. Further, in earlier systems such as iCAP [6], such concepts are simply ignored and left out of the construction. In another direction, bottom-up approaches allow for reusing and composing context features [21], but creators can become stuck when they do not know a priori what context features may be useful for expressing an abstract idea they have. To overcome the shortcomings of top-down and bottom-up approaches, we draw on theories from opportunistic planning [12] to support an *opportunistic* construction process, in which a creator can follow both top-down or bottom-up processes at any time. Further, decisions and observations during construction may suggest new ideas or illuminate problems that cause the creator to shift their strategy.

Beyond opportunistic construction, Affinder provides easy ways to forage for context features that might be relevant to a user's conception of a situation (e.g., what objects might be available to interact with; what atmosphere or feelings the situation might evoke). The naive approach of scanning through hierarchies of context features doesn't work when some hierarchies, like place categories obtained from the Yelp Places API, are thousands or more items long. Simple text search can help, but only if the user knows the keywords matching the context features, and what context features may be useful for expressing particular intermediate concepts. Overcoming these shortcomings, Affinder uses textual metadata from a Yelp API to create an unlimited vocabulary [10] of terms associated with the context features that users can query for based on their conception.

Social and Crowd Computing

The opportunistic nature of OCEs call for programming environments that allow experience creators to directly express the conditions that people should be in for them to participate in an OCE. This is unlike existing programming environments for social and crowd computing [18, 15, 19, 2, 23], which focus on supporting the expression of structures through which tasks can be completed (e.g., workflows and organizational structures) but largely ignore the changing conditions of the participants. Systems such as Jabberwocky [2] and Foundry [23] support expressing richer notions of identity, expertise, reputation, and social relationships, but retain the underlying assumption shared by prior systems that workers can be recruited at will to fulfill roles. In contrast, Cerebro assumes by design that OCEs can only meaningfully be triggered when the appropriate situations arise, and even then, that some interactions may not be realized if people decide not to participate or are actually not available.

As a consequence, executing an OCEs requires an opportunistic run-time engine that constantly monitors for the changing conditions of people and the OCE itself, and that opportunistically match available resources (i.e., our participants in certain situations) to needs (i.e., opportunities suggested by the OCE) as situations arise. Unlike existing solutions for programming crowd workers that assume that specific (human) resources can be recruited for meeting the specific needs of the system on-demand (e.g., [4, 3, 16]), our approach waits for resources to become available at run-time and makes no a priori assumptions about how or if certain needs will be fulfilled. Viewed this way, our approach bears a resemblance to concepts such as futures and promises [17, 11] in machine programming languages for optimizing programs based on resource availability, but in our setting is used not for optimization but for enabling opportunistic interactions at a distance.

Existing social applications already make use of people's shared context for supporting opportunistic interactions when people are in the same place or doing activities at the same time. For instance, Yik Yak and Snapchat Campus Stories promote conversations among people who frequent the same places, and Strava automatically groups overlapping riding routes that occur over the same time. Cerebro is designed to support a variety of interaction structures that extend beyond the same time, place, and activity models used in most current applications. Further, Cerebro provides general supports for programming opportunistic interactions that abstracts away the low-level details of tracking situations and coordinating interactions.

OVERVIEW OF OCEs

We envision the creation of Opportunistic Collective Experiences (OCEs): programs that recognize and facilitate opportunistic interactions. In this section, we describe how experience creators may use OCEs to express rich situations for opportunistic interaction, in ways that apply to geographically distributed contexts; and implement a variety of interaction structures that facilitate shared collective experiences across time and distance. In the following sections, we describe how Affinder and the CE API can be used to support these goals.

Expressing Situations

OCEs are designed for interaction; describing the situations in which these interactions might be feasible thus requires expressing the aspects of the situations that might enable an interaction. Table 1 highlight example aspects, including the actions that can be taken (e.g., jump over puddles), the objects to interact with (e.g., castles-like buildings), the activity or state of individuals (e.g., waiting in line), and the feelings a situation evokes (e.g., feeling peaceful while watching a sunset reflect off the water). Writing OCEs requires finding ways to express such aspects of a situation in ways that machine can understand. For instance, an action like jumping over puddles might be represented with intermediate concepts such as 'raining' and 'places with asphalt,' for which we might find matching context features based on a weather API and with Yelp location categories (e.g., parking lots and playgrounds)

For OCEs to connect people across geographically distributed contexts, situational descriptors should attempt to cover as many places as possible. We can help creators identify all the places or situations that match a concept, and to support their finding related context features. For example, in an OCE

Aspects of Situations	Example Situations	Concepts (matching context features)
Actions that can be taken	Jumping over puddles	Rain and asphalt (playground or parking lot)
Objects to interact with	Castle-like buildings	Castles, model-sized castles (mini-golf), or old Gothic architecture (churches or historical building)
On-going activities	Waiting in line	Waiting for fun (water parks or concert venues) or waiting for transit (airport lounges or public transit)
Feelings that can be evoked	Feeling peaceful while watching a sunset reflect off the water	Sunset time and near water (marinas, lakes, or beaches)

Table 1. Aspects of situations and the ways we might want to represent them as machine detectable context features

for writing a storybook together that contain a scene about standing proudly in front of a castle, we might want to help designers find castle-like buildings by connecting to context features of places such as castles, churches, historical buildings, and so on. We can also help creators recognize that the concept may be 'stretched,' or realized in other ways. For instance, creators may find situations with scale-models of castles (e.g., on the mini-golf golf) to be completely appropriate for the experience, or even preferred.

Interaction Structures

We introduce and motivate in this section four different structures for opportunistic interactions at a distance, that we wish to support when writing OCEs (Figure 2):

- *Same situation, same time* connects participants who are not co-located through their situational context. For example, "Virtual Bumped" turns waiting in line by yourself into a shared opportunity to say hello to an old friend who's also waiting in line somewhere else in the world.
- *Same situation, across time* creates links between an individual's current situations and people who were recently in the same situation. When a participant takes a photo in "Sunrise to Sunset", they get to see how their photo fits into the collective timelapse of all the other people who've stop to appreciate tonight's sunset.
- *Unique contributions towards a shared goal* brings people who are in different situations together through a shared activity with a common goal. For example, a person in a parking lot with puddles and a friend in a nature preserve with trees can both contribute in unique ways to unfinished goals in a collective "Parkour Challenge."
- *Participation influences future interactions* creates a coherent chain of interactions based on prior contributions. For example, in "Storybook" a group composes a story by opportunistically contributing both photos from their surroundings that relate to the previous sentence of the story, and a new sentence to move the story along.

AFFINDER

We first focus on the challenge of representing rich situations in ways that can be automatically detected across a range of geographically distributed contexts. For this we introduce Affinder, a block-based programming environment for expressing situation detectors. Affinder supports (1) opportunistic construction of situations concepts from context features through

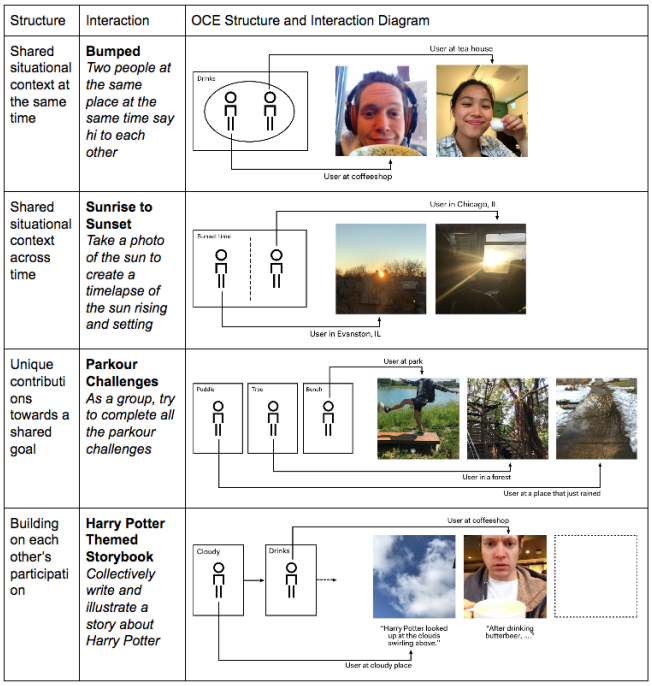

Figure 2. OCEs can support a variety of interaction structures that connect people through shared contexts, goals, and actions. This figure presents four such structures with associated examples of OCEs.

a single, visual workspace and (2) stretching situation concepts to many geographic contexts through a search interface.

A Visual Workspace for Construction

Creators interact with Affinder's visual workspace to *declare concepts*, *forage for features*, and *compose representations*, all in an opportunistic fashion; see Figure 3. The interface provides visual blocks that represent concepts, context features, and the logical operators that connect them. These visual blocks are surfaced through Affinder's interface, which consists of (1) the *Toolbar*, which is a set of tabs that expand to panels that organize the building blocks; (2) the *Work Area* which is where building blocks can be dragged and the final detector is composed; and (3) the *Search Interface* for finding features using text queries.

Declaring concepts entails breaking down an idea about a situation into smaller concepts. These smaller concepts serve as a link between a conceptually-abstract situation and the available context features, and can be declared explicitly with *concept variables*. A creator can declare concept variables

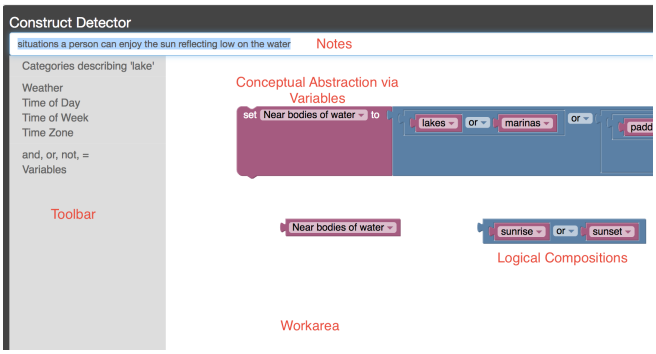

Figure 3. Affinder's toolbar provides access to building blocks such as context features derived from weather, time, place in addition to logical operators like *and, or, not,* =. The work area is used to store relevant features and compose representations from building blocks.

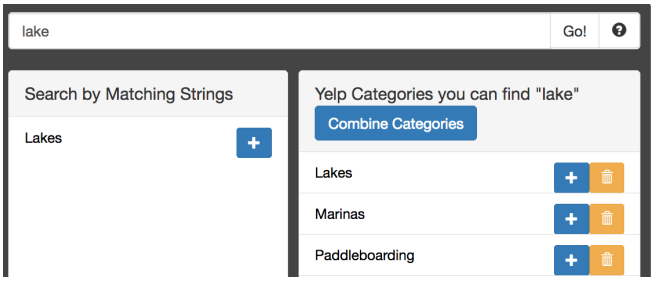

Figure 4. Affinder's search interface returns relevant place category features. A button next to each "adds" the feature to the toolbar. A set of features can be combined together too: (1) irrelevant place features can be "trashed", while the remaining places can be "combined", concatenating the remaining items into a multiple-value *or* statement.

before defining their contents; this provides a visual reminder to compose building blocks later for the concept. High level ideas can also be recorded in a *Notes* area, that can be easily referenced or edited as decisions and observations during construction suggest new concepts or reveal problems.

Forage for features involves both searching and browsing for building blocks. Creators use their declared concepts to guide the types of building blocks they might look for. A creator can use the *Toolbar* to navigate to building blocks based on categories such as weather, time of day, time of week, and time zone (Figure 3, Left). Creators can also use the *Search Interface* (Figure 4) to find contextual features based on a thousand place categories on Yelp. Instead of restricting a creator to search directly on the keywords that match these place categories, the search engine supports a creator typing arbitrary queries that are matched to Yelp reviews in the various place categories, based on which the search results are ranked (see Implementation section for more details). This allows a creator to query for aspects of a situation like the ones shown in Table 1.

For example: (1) the action 'jumping' may match the category 'Playground' through a review that says "[The *playground*] had so many slides and *jumping* things and a huge ball pit;" (2) places with 'castles' to interact with may match 'Mini-golf' through a review that says "The classic *mini-golf* course... [has] a cascading creek with water wheel, huge *castle*, bridges..."; (3) the act of 'waiting in line' may match 'Department of Motor Vehicles' through a review that says "This is the [*Department of Motor Vehicles*] so of course you'll have to *wait in line*"; (4) the atmosphere or feeling of being in a 'beautiful' place may match 'botanical gardens' through a review that says "come maybe half an hour before the sun really starts setting and then be amazed as *beautiful* shadows are cast throughout the *botanical garden*."

Composing Representations starts after creators find several context building-blocks which they combine together with logical operator blocks like *and, or, not,* =. Figure 3 shows an example of a logical composition to represent situations that involve either sunsets or sunrises.

As an example scenario for using Affinder, consider a creator who wishes to express situations for the "Parkour Challenges" OCE. A creator records actions for parkour that come to mind, such as jumping, climbing, or vaulting. While browsing the weather features in the toolbar, they stumble upon "rain," which reminds them of puddles, which could be jumped over. Puddles often form on uneven asphalt, so they use the search interface to find places likely to have asphalt, discovering 'Parking Lots' and 'Playgrounds.' They can then use an 'AND' operator to combine these two concepts to create a "situations good for puddle jumping" detector.

OPPORTUNISTIC COLLECTIVE EXPERIENCE API

In this section, we detail our API, which enables experience creators to write concise OCE programs. The API allows for expressing programs at the level of interaction structure and abstracts away details at levels below such as tracking and coordinating users across situations. The API supports writing an OCE using three programming structures: (1) *needs*, which describe situation-specific interactions, and (2) *callbacks*, which are functions that trigger on specific conditions to update the OCE, and (3) *templates*, which determine what UI to load in the mobile app for participants. We describe each component below.

Needs

Needs enable creators to specify how individual interactions come together to create an OCE. Each need is a single interaction and its required situation. A need's description includes (1) *needName*, which is used to match to a user-facing template for interaction; (2) *situation*, which links the need to a situation detector from Affinder, and specifies the number of users who need to be in the described situation at the same time for the interaction to occur; and (3) *numberNeeded*, which specifies how many users should participate in the OCE before it is complete. For example, the "Sunrise to Sunset" OCE (from Figure 2) code below seeks 20 participants to take photos when they happen to see the sun reflecting off the water:

```
{
   "needName": "sunsetPhoto",
   "situation": {
      "detector": Affinder.lookup("sun reflecting off
          the water"),
      "number": 1
   },
   "passToTemplate": {
      "instruction": "Take a photo of sunlight!"
```

```
    },
    "numberNeeded": 20
}
```

Beyond expressing the same *Same Situation, Across time* interaction structure shown in this example, the Needs structure can easily be used to also express a *Same Situation, Same Time* interaction structure by changing the number of people for the situation (e.g., 2 people for Virtual Bump). We can also express the *Unique contributions towards a shared goal* interaction structure, simply by adding multiple needs, one for each type of contribution desired.

Callbacks

Callbacks are used to conditionally update an OCE's needs based on the state of the OCE through (1) the *trigger*, which is a boolean expression; and (2) a *callback function* that runs when the trigger is true, with the most recent participation passed in as an argument. With these, we can express the *Participation influences future interactions* interaction structure, as in the code below from the OCE "Storybook" (from Figure 2):

```
{
    "trigger": "newSubmission() && (
         numberOfSubmissions() <= 7)",
    "callbackFunction": function(newSubmission) {
        let newNeed = {
            "needName": page + i,
            "situation": {
                "detector": Affinder.lookup(
                     newSubmission.content.
                     nextSituation),
                "number": 1
            },
         ...
         };
        addNeed(experienceId, newNeed);
    }
}
```

The use of callbacks allow for composing opportunistic interactions for creating richer experiences with multiple stages, and those that combine multiple interaction structures into one OCE. For example, we could have a multi-level "Parkour Challenge" (from Figure 2), where once all the current challenges are completed, a callback can unlock advanced challenges that require multiple people to complete actions at the same time.

Templates

Templates describe the UI layout that participants see on their mobile devices during and after they participate in an OCE. Templates can contain blanks that are filled in with additional information provided from the need. With templates, we can quickly build new OCEs by re-using interface components that can still be further customized. For example, "Sunrise to Sunset" and "Bumped" both use the same `uploadPhoto` interface filled in with custom instructions from the need.

OPPORTUNISTIC INTERACTION ENGINE

Running OCEs requires an engine that continually checks for opportunities for interaction, and executes them when possible. This run-time engine must track changing resources (users) and tasks (needs), contextualize how individuals' user context maps to needs, and attempt to coordinate interactions between the two. We thus present the Opportunistic Interaction Engine (Figure 5), which consists of:

1. The *User Monitor*, which maintains and monitors the locations of users, and updates their context as their location and situation changes.

2. The *Opportunistic Coordinator*, which actively identifies which users could fulfill which needs, strategizes on which users to notify for which needs, executes and monitors the strategy, and re-strategizes if unsuccessful or a change in users or OCEs is detected.

3. The *OCE Manager* holds OCE programs, tracks their associated current needs, and handles the progression of OCEs based on their defined callbacks.

Execution Flow

The engine begins execution when a user's location changes. The Location Tracker (1) captures location updates and uses the Context Translator (2) to label the location data with context features, including location categories (e.g., from the Yelp API) and current weather conditions (e.g. from the Open Weather API). These User Contexts, along with Current Needs from the OCE Manager, are passed along to the Opportunistic Coordinator (3).

The Opportunistic Coordinator begins by identifying what needs a user is eligible for (4). The Identifier evaluates the Situation Detectors for the current needs using the user's context features in the boolean expression. The Strategizer, now knowing which users are eligible for which needs, begins to strategize how to fulfill the needs (5). This step considers how many users are needed to complete a need, including if multiple users are needed synchronously. If a strategy is found that can match the required users to a need, the Executor runs and monitors that strategy for the need (6), notifying users to participate (7). When the users opens the notification, the front-end application loads the appropriate template based on the OCE and the need the user is assigned.

As the strategy executes, the Strategizer actively considers whether the strategy may need to be revised. Strategies are revised when either a change in resource is detected or the current strategy is not receiving participation. If users locations or current needs change, the system re-runs (2) through (5) to form a new strategy. If progress stalls, the Strategizer can re-strategize how to progress by removing the current user matched with the need and finding a new user, if possible.

When the Executor receives participations (8), it checks if the strategy is completed (9) and sends the data to the Progressor in the Experience Manager (10). The Progressor fetches the Callbacks for the OCE (11) and executes those ready to run based on the OCE's current state. The Callback is used to add or remove needs from the current needs based on results from participation (12). Upon update of the current needs, the Identifier detects a change in resources and begins the process of strategizing to fulfill needs once again.

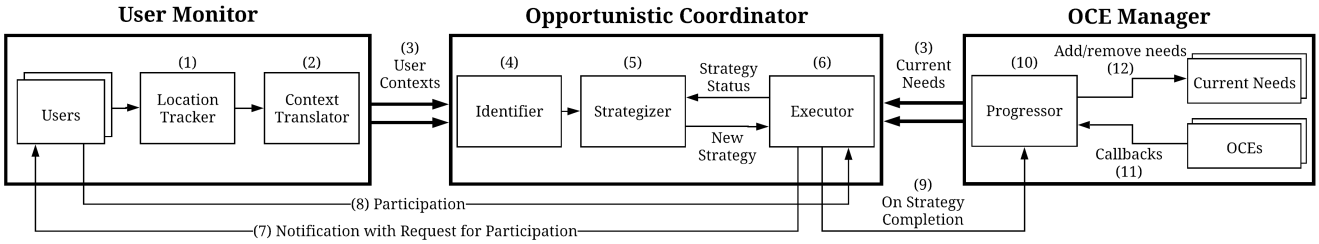

Figure 5. The Opportunistic Interaction Engine allows for real-time matching between users and their contexts to OCE needs. The *User Monitor* constantly monitors users' locations and updates their context features. User contexts and the current needs of the OCEs are sent to the *Opportunistic Coordinator*, where a strategy for opportunistically matching users to needs is created and executed. The Opportunistic Coordinator both monitors the execution of the strategy and listens for new user contexts, re-strategizing if needed. On strategy completion, the OCE Manager updates the current needs based on the OCE's callbacks and strategy output.

IMPLEMENTATION

Cerebro consists of a client application written in Cordova for iOS, and a Meteor.js and MongoDB backend. The iOS front-end handles geolocation tracking, and the presentation of notifications for participation in OCEs. Our backend has three components: (1) Affinder, a Meteor.js application that uses Blockly [9] for its interface. The search engine on place categories was built by applying TF-IDF to a corpus of 1241 documents corresponding to Yelp place categories. Each document comprised reviews for all listed places in 11 major metropolitan areas [14]. Document relevance for a query was based on the sum of TF-IDF values for all terms in a query [20], and the 25 top place category were returned. (2) OCE API, a series of high level functions built in Meteor.js; and (3) Opportunistic Interaction Engine, a second Meteor.js application that handles the execution of OCEs and communications with the Context Translator, another Flask application.

AFFINDER USER STUDY

The goal of the Affinder study is to understand how Affinder supports the expression of rich situations for OCEs. We aim to answer the following research questions:

RQ1 To what extent can Affinder support creators of OCEs in expressing conceptually rich situations in a representation that computers can understand?

RQ2 To what extent can Affinder support creators of OCEs to develop representations that can be applied to multiple instances of context distributed across different environments and times?

Method and Analysis

Participants

We recruited 7 (M=5, F=2) undergraduate students from a mid-sized U.S. university for a 1 hour lab study. Participant age ranged from 19 to 22 ($\mu = 20.40$, $\sigma = 1.14$). All participants had some prior programming experience, and understood how context features like local weather and place categories could be obtained from location-data. They were compensated $20.

Procedure

Participants were asked to use Affinder in a lab setting for 1 hour, while one of the authors observed. The first 10 minutes were used to teach users how to use Affinder by demonstrating the construction process through a toy example. In the next 40 minutes, participants were given situation prompts and were asked to create a detector such that (1) the situation is more likely than not to be occuring in a context; and (2) the detector will recognize as many users that are in this situation. Example prompts included: viewing sunlight on the water; exercising on a warm day in spring; performing maneuvers for parkour; situations awesome to throw a paper airplane. Participants were asked to vocalize their thought process while constructing detectors, and to complete as many detectors as possible in the timeframe. The final 10 minutes consisted of a post-interview where participants were asked to recall and describe strategies for expressing situation detectors. All participants consented to an audio and screen recording.

Measures and Analysis

To understand how Affinder's interface supports the creation of rich situation detectors, we refer to observations during the study and post-study interviews that ask users to recount their strategies. We record (1) ways they interacted with interface features and (2) how their processes were opportunistic i.e. if plans changed during construction.

We investigate how Affinder supports the expression of representations that generalize across different places. We note if Affinder's search interface helped participants discover places they had not thought of earlier, through observing if results seen during the process inspired new concepts or if participants said "I wasn't thinking about [places like X] before."

Results

Affinder supported potential creators in (1) aligning their user models of a situation idea with the context features as close as possible and (2) stretching those into representations that the system could automatically detect across a variety of geographically distributed contexts. This section addresses our research questions with results from user observations, talk-alouds, and questioning during user testing.

RQ1: Expressing Situations from Context Features

Our 7 participants constructed a total of 26 situation detectors in response to 13 prompts. An example construction expresses "awesome to throw a paper airplane" as open areas without

many people (disc golf, parks, playgrounds, mobile home parks, hiking), weather is not disruptive (not windy, clear, or clouds), and while there is daylight. Another example expresses "people are enjoying winter" as fun winter activities (ski resorts, ski schools, fondue, coffee and tea, coffeeshops) and winter weather (snow or cold). Another example expresses "start a snowball or food fight" as anywhere there is snow (snow) or school cafeterias (cafeteria, colleges and universities, elementary schools, middle schools, and high schools).

In doing so, participants created a variety of concept variables to help them express their idea into smaller concepts. P3 used concept variables to declare concepts he would pursue later: *'city parkour' and 'nature parkour'*. The toolbar gave users access to features of time and weather. P5 expressed "situations where old friends might catch up over drinks" as casual drink places (coffeeshop or brewerie), during times for hanging out (Friday night or Saturday).

Participants used a mix of bottom-up and top-down strategies. 3 of the 7 participants went top-down by explicitly declaring concepts before starting to forage for features. P1 used Notes to list every concept for a 'sunset over water' situation: "swimming pools, bodies of water (oceans, lakes, rivers, ponds, gulfs), airplanes, boats, tall buildings in cities next to water, anytime between 6am and 8pm, sunny (windy?)." The remaining 4 participants started bottom-up, defining one aspect of the situation before rethinking about other concepts.

We saw evidence for opportunistic construction, where participants switched strategies. This often occurred when participants realized a detector did not exist for a concept. For example, P5 expected a feature for labeling a restaurant as expensive; she compromised and instead choose categories like Modern European and Wine Bars to approximate 'expensive.' Similarly, P7 discovered that many green spaces were not labeled as parks, which was problematic for finding many places where someone might be relaxing in warm weather. They adjusted by foraging for other categories, such as Disc Golf, Fishing, and Playgrounds.

RQ2: Stretching to General Representations

All 7 participants leveraged the search engine to forage for context features from the Yelp Places API that matched their initial concepts, and to stretch their concepts based on other places found. For example, P1 tried to represent "bodies of water." By searching for terms such as oceans and lakes, they were able to find many place features matching their concept (Lakes, Beaches, Marinas, Diving). P6 said: "I liked to see other examples of other [place categories], I didn't have to think about all these other [place categories] on my own."

Furthermore, the search engine inspired users to think about places that were conceptually different than their original concept. We present some examples from our user study where the search engine supported stretching concepts (see Table 2). Participants updated their concepts when they saw search items different than their original mental image, helping enrich their understanding of the nuances of which types of contexts are applicable to the situation being expressed. For instance, this allowed users to recognize Mini-golf as a possible location

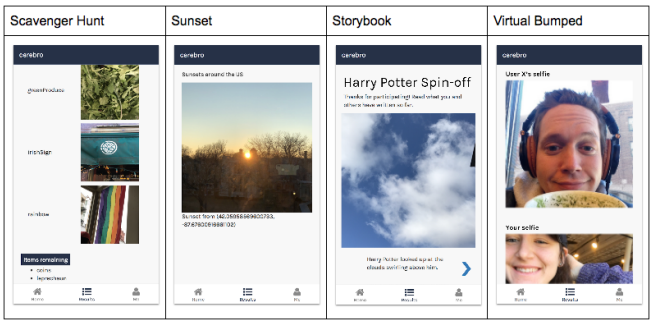

Figure 6. Sample screenshots from the results pages for all OCEs run in the study. From left to right, these show (1) three items that participants collected during a St. Patrick's day themed scavenger hunt, (2) a single frame from the sunset timelapse, (3) a page from the storybook, where the situation chosen for illustrating the page was "swirling clouds", and (4) two selfies from people who virtually bumped at coffee shops.

for 'castles,' and think of Fishing and Disc Golf as locations for being active beyond parks in the city. Search also allowed users to find inappropriate contexts that they initially would have included. For instance, a user was originally thinking of Parks as a category for throwing paper airplanes, but recognized through search that some Parks, such as Dog Parks would more likely lead to "a dog chewing up the airplane."

OCE EXPLORATORY USER STUDY

We conducted an OCE exploratory user study that aims to answer the following research question:

RQ1: How do people interact with OCEs through the four interaction structures describe in Figure 2?

Method and Analysis

Participants

We recruited 32 participants (M=17, F=15) who were either students or recent alumni of the research lab, or their family members. Participant age ranged from 17 to 31 years old ($\mu = 22.16$, $\sigma = 3.18$). The study duration was 2 weeks, with participants volunteering to install the application without any compensation. Post-study interviews were conducted on a subset of the participants who received an above average number of notifications, for which those users were compensated $10.

Procedure

To test OCEs, we built an iPhone application called OCE-Platform that communicates with the Opportunistic Interaction Engine. OCE-Platform acts as the front-end interface and the Location Tracker for the Opportunistic Interaction Engine (see Figure 5). When the user's location changes and their situation matches an OCE need, they are sent a notification to participate. Next, OCE-Platform loads a participate UI based on the OCE template and matched need, prompting the user to complete the interaction for the need. After participating, OCE-Platform loads a results page UI for the OCE that combines the interactions of all participants (Figure 6).

Participants were instructed to install and use OCE-Platform for 2 weeks in their daily lives, participating when able. When notified to participate in an experience, they were told what to contribute in order to satisfy the need presented. At the

Search Query	Initial Concept	Place Category That Inspired Updates	Updated Mental Image or Concept
Castles	Historic castles, gothic architecture, and old churches.	Mini-Golf	Mini-Golf often has a scale-model of a castle.
Parks	Throwing a paper airplane in park, field, large open spaces.	Dog Park Skate Park	Dogs might eat paper airplane. Skaters would be disrupted. Places with not many people/animals moving around.
Balcony	An apartment balcony for high places	Opera houses	Large Performance Halls

Table 2. Examples of how search terms for initial concepts surface items not thought of and that update the users concepts for a situation

end of the study, participants who received an above average number of notifications were asked to participate in a post-study interview.

We ran four experiences during the study. This includes Storybook, Sunset, and Virtual Bump from Table 2, and a modified version of Parkour Challenge called Scavenger Hunt, where participants collect photos of items on a list as challenges. All experiences except Sunset were re-run every 2-3 days once all experience needs were close to fulfilled.

Measures and Analysis

To understand how people participate and interact with OCEs, we consider the participation rate obtained from log data, and use the post-study interview to understand why participants may have liked OCEs. In the post-study interview, we specifically asked participants to elaborate on what they felt when notified and when participating in the experience.

In pre-analysis, we removed 3 participants who, due to technical reasons, were not able to receive notifications during the study. Of the remaining 29 participants, we conducted post-study interviews with 10 participants.

Results

OCE-Platform enables opportunistic interaction to occur between participants across distance and time. 75% of participants contributed to experiences across 3 different U.S. states during the study. The average response rate to notifications for participation was 37.5%. Participants received an average of 9 notifications ($\sigma = 9.00$) and participated an average of 3.6 times ($\sigma = 5.13$). While most participation led to successful interactions, in some cases participants were asked to contribute when they were not in the required situation. Some users chose to still contribute to the need even though their situation did not match. We return to these points in limitations.

OCE-Platform further enables a variety of interactions to occur, including those that require synchronous participation (summarized in Table 3). Most OCEs have similar participation rates, indicating the feasibility of both synchronous and asynchronous OCEs. However, we find that Storybook had a significantly lower participation rate than other OCEs. We hypothesize this is because the needs were specific enough that, if the system notified a user for the situation incorrectly, participants would not be able to illustrate the sentence.

When participating in experiences, participants were aware of how their current situation overlapped with others across distance and time. Two users "bumped" at different coffee shops, with one saying: *"I knew that we were doing the same thing at the same time, and I thought that was cool."* In the case of asynchronous interactions like Sunset, a user said it made them think, *"all the people using the app are in the same community as me" and "we're all looking at the same thing"*, even though the interaction occurred at various times.

The automatic recognition and facilitation of interactions by OCE-Platform allowed users to feel that there were more opportunities to share moment with others. For example, two users "bumped" while at grocery stores in different states. One said, *"I wouldn't have any other reason to see or say 'Hi' to [other user], so I really treasure the fact that I got to bump into her."*. Moreover, a user was excited that their sibling was also using OCE-Platform because they usually only talk if they see something that reminds them of each other, a rare occurrence: *"I thought it would a fun little activity for us to do because we are all super busy and just being able to share with my brothers would be a lot of fun."*

Participants also felt that OCE-Platform helped lower the barriers to interacting with others since the opportunities were low-effort moments of interaction. For example, a user said: *"I don't have an hour to talk to this person...but can I spare a few minutes to say hello? Yes I can."* Further, some participants found that the momentary nature of the experience reduces the "anxiety" of worrying if someone will respond: *"made it less about communicating and more about participating... in this community and in this activity"*.

OCE-Platform further promoted interaction outside of OCEs as it gave a shared moment to those who may not have interacted otherwise. For example, a user said: *"Because I virtually bumped into [another user] twice, I really considered reaching out to her but didn't because this week I was really really busy. But I seriously considered that."*

Participants felt excited by opportunities to participate and were eager to contribute the best they could. For example, a user was asked to find the moon for a Scavenger Hunt OCE: *"I got the [notification] and I was very excited... We looked up, but we couldn't find the moon anyway, so we ran around the building [trying to find it]"*. In another case, a user successfully took a picture of the sky for a Sunset OCE by going to the roof of their building, citing they did so because they were excited to contribute to the group.

Finally, participants became invested in the completion of the OCE and the shared artifact it created. A user participated in

OCE	Notifications	Participations	Participations from Unique Users	Participation Rate
Virtual Bumped x3	142	68	27	47.90%
Sunset x1	18	8	8	44.40%
Storybook x4	60	10	7	16.70%
Scavenger Hunt x3	36	20	13	55.50%

Table 3. Participation across OCEs. Each Virtual Bumped OCE allowed for many users to bump during each run, resulting in the significantly higher notification count.

Scavenger Hunt and kept checking the results to see if anyone had fulfilled, "Lake". They even went as far as to tell their friends to go to the lake to try and complete it.

Study Limitations

We found that our system had a high number of false positives when recognizing situations due to inaccuracies in location detection and Yelp API place locations. This most frequently occurred when a user was in a downtown area, multi-story building, or train. In each, the density of businesses or the quickly changing location caused the system to inaccurately recognize the user's context. If a user received a notification for an incorrect situation they could not receive another notification for the next hour (implemented to prevent notification spam), which could have caused them to miss an opportunity to properly contribute.

Our study ran with a set of users who all knew each other, allowing us to learn about opportunistic interactions within a community; however, we don't know how these initial findings would translate if participants did not know each other as well. Additionally, the majority of our users frequently interact in-person in addition to interactions on the OCE platform, which likely influenced the amount people interacted with OCEs and their motivations.

DISCUSSION

Having demonstrated the potential for Cerebro to support opportunistic interactions in which people experience shared engagement without explicit planning, we revisit elements of Cerebro's design that may in general facilitate *computer-facilitated opportunistic interactions*.

Write programs that react to people's changing situations

Just as reactive programming enables UIs to constantly update with the newest information, we can write programs where the people in the program automatically change as their situations change and opportunities occur in different places. This new type of programming — where lines of code monitor the user pool, try to run when opportunities occur, and continue trying until they receive a response — allows us to write code with opportunistic interactions embedded.

Interaction Structures Enable Connection Across Distance

By thinking about interaction structures that link together individual situations, we can create opportunities for shared engagement across time and distance. Cerebro makes this possible by automatically recognizing moments for interactions we would of otherwise have missed and enabling designing new types of interaction structures for these moments.

Conceptual Stretch uncovers new opportunities for interaction

By recognizing that we can describe opportunities across a myriad of situations, we expand the range of situations we can include in an opportunity definition. This increases the likelihood of an interaction without diluting it.

Opportunistic Planning supports aligns concepts and features

Affinder's single, visual workspace aided opportunistic strategies that relied on re-referencing high-level concepts in between defining the smaller components for various concepts. This enabled for easily adjusting high-level concepts when alignment between a concept and context features cannot be found, and helped prevent creators from getting stuck.

FUTURE WORK

Based on this work, we present directions for future work:

Opportunities for Advanced OCEs

In our vision for opportunistic interactions, our API will expand to (1) support a wider range of opportunities including specifying a set of synchronous needs and be able to specify participants must be collocated or share a given relationship, (2) allow for protecting OCEs from getting stuck by choosing a more general version of a situation to expand to if no one can be found in the original situation, (3) support longer term experiences through using participation history in OCE structures or situations. These advances would allow for OCEs such as spontaneous flash mobs in a public plaza with dancers and music automatically coordinated by the system.

Forming New Connections and Creating Micro-Communities

We want to explore principles and methods for connecting individuals through multiple collective experiences and moments across time. We look towards ideas of relational continuity [7] and social networking through "everyday encounters" [8, 24, 25] as inspiration for modes to form new connections and micro-communities through chained experiences. For example, we could have an OCE for a ramen slurping competition. The next week, the OCE could ask everyone who previously participated if they wanted to slurp ramen together again. This could start off as an opportunistic interaction, but grow to be a habit beyond the app where the same people go weekly at the same time to eat at different ramen shops around the world.

Advanced Coordination Algorithms

In regards to the Opportunistic Engine, there remains a lot of work in algorithms for best matching participants and OCE needs. For example, we could have an algorithm that strategizes across all users when determining which OCE to notify for if they are eligible for multiple. Such an algorithm might account for how rare or temporal an opportunity is and predict

what opportunities are likely to occur in the future based on user movements, opportunity history and time of day.

Richer Development Environment for Detectors

Just as developing environments are important for programming with crowdworkers, additionally support for Affinder could provide feedback on accuracy and reach of detectors using participation history and machine learning. These tools are especially important as we think about OCEs across larger, more distributed users bases.

REFERENCES

1. Gregory D Abowd and Elizabeth D Mynatt. 2000. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 1 (2000), 29–58.
2. Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. 2011. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 53–64.
3. Michael S Bernstein, Joel Brandt, Robert C Miller, and David R Karger. 2011. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 33–42.
4. Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and others. 2010. VizWiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 333–342.
5. Anind K Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. a CAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 33–40.
6. Anind K Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive prototyping of context-aware applications. In *International Conference on Pervasive Computing*. Springer, 254–271.
7. Steve Duck, Deborah J Rutt, Margaret Hoy, and Hurst Heather Strejc. 1991. Some evident truths about conversations in everyday relationships all communications are not created equal. *Human communication research* 18, 2 (1991), 228–267.
8. Mattias Esbjörnsson, Oskar Juhlin, and Mattias Östergen. 2003. Motorcycling and social interaction: design for the enjoyment of brief traffic encounters. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*. ACM, 85–94.
9. Neil Fraser. 2015. Ten things we've learned from Blockly. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*. IEEE, 49–50.
10. George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. 1987. The vocabulary problem in human-system communication. *Commun. ACM* 30, 11 (1987), 964–971.
11. Robert H Halstead Jr. 1985. Multilisp: A language for concurrent symbolic computation. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7, 4 (1985), 501–538.
12. Barbara Hayes-Roth and Frederick Hayes-Roth. 1979. A cognitive model of planning. *Cognitive science* 3, 4 (1979), 275–310.
13. Scott Hudson, James Fogarty, Christopher Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny Lee, and Jie Yang. 2003. Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 257–264.
14. Yelp Inc. 2018. Yelp Fusion API. `https://www.yelp.com/developers/documentation/v3`. (2018).
15. Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. 2011. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 43–52.
16. Walter S Lasecki, Raja Kushalnagar, and Jeffrey P Bigham. 2014. Legion scribe: real-time captioning by non-experts. In *Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility*. ACM, 303–304.
17. Barbara Liskov and Liuba Shrira. 1988. *Promises: linguistic support for efficient asynchronous procedure calls in distributed systems*. Vol. 23. ACM.
18. Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. 2009. Turkit: tools for iterative tasks on mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*. ACM, 29–30.
19. Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z Gajos. 2011. Platemate: crowdsourcing nutritional analysis from food photographs. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 1–12.
20. Juan Ramos and others. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. 133–142.
21. Daniel Salber, Anind K Dey, and Gregory D Abowd. 1999. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 434–441.
22. Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman. 2014. Practical trigger-action

programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 803–812.

23. Melissa A Valentine, Daniela Retelny, Alexandra To, Negar Rahmati, Tulsee Doshi, and Michael S Bernstein. 2017. Flash Organizations: Crowdsourcing Complex Work by Structuring Crowds As Organizations. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 3523–3537.

24. Bin Xu, Pamara Chang, Christopher L Welker, Natalya N Bazarova, and Dan Cosley. 2016. Automatic archiving versus default deletion: what Snapchat tells us about ephemerality in design. In *Proceedings of the 19th ACM conference on computer-supported cooperative work & social computing*. ACM, 1662–1675.

25. Bin Xu, Alvin Chin, Hao Wang, Lele Chang, Ke Zhang, Fangxi Yin, Hao Wang, and Li Zhang. 2011. Physical proximity and online user behaviour in an indoor mobile social networking application. In *Proceedings of the 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. IEEE Computer Society, 273–282.