Bluewave: Enabling Opportunistic Context Sharing *via* Bluetooth Device Names

1st Author Name
Affiliation
City, Country
e-mail address

2nd Author Name
Affiliation
City, Country
e-mail address

3rd Author Name
Affiliation
City, Country
e-mail address

ABSTRACT
Context-aware applications oftentimes require devices to share information with each other in order to provide users with relevant information and services. However, current context-sharing techniques require significant amounts of setup before they can be used, making them cumbersome when devices need to share information once or spontaneously. To address this problem, we present Bluewave, a lightweight but extensible technique that allows devices to *opportunistically* share context when they are nearby. With Bluewave, devices upload their context to the cloud, and modify their Bluetooth friendly name to contain a URL to this information. Other devices can then scan for these links and download the context without having to know of each other or pair in advance. Bluewave provides a simple way for users to share context through their mobile devices, which is useful for applications where the *environment* needs to collect information about the *user*. Additionally, our system's design has been guided by user feedback, and integrates privacy controls to let users manage how their context is shared. In this paper, we describe Bluewave's architecture, and show how developers can use our system to create a wide range of context-aware applications. We argue that Bluewave's low battery consumption, combined with its speed and compatibility with existing mobile devices, makes it preferable to comparable techniques.

Author Keywords
Ubiquitous computing; mobile interaction

ACM Classification Keywords
D.3.3 [Language Constructs and Features]: Frameworks

INTRODUCTION
Context-aware computing has the potential to improve our lives by allowing computers to provide us with the right information at the right time. Yet despite over two decades of research following Weiser's vision [38,39], there are still many types of context-aware systems that are conceptually simple, but difficult to create because they rely on information that they cannot obtain on their own. Consider, for example, a public display that can intelligently translate its contents because it knows the language preferences of nearby users, or a store that can tell arriving customers which aisles they need to visit because it knows the contents of their shopping list. For these systems to become pervasive, there needs to be a way for users to quickly and easily share some context (*e.g.,* language preferences and shopping lists, respectively) with their immediate environment. Current context-sharing techniques, however, have usability issues, by requiring devices to share credentials, pair, or install and run a dedicated application before they can share information, making them impractical for short, one-time exchanges of information like the ones described above. Consequently, while these types of systems are *technically* possible [41] there is still a need for technologies that can make them practical.

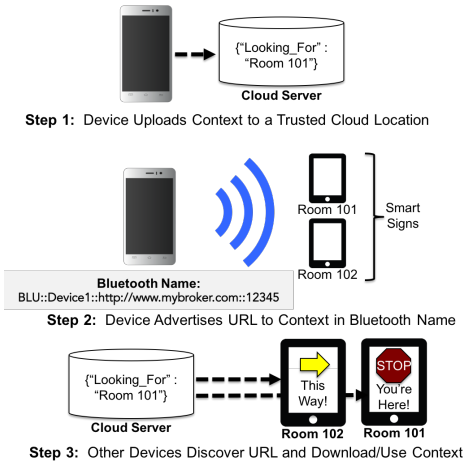


Figure 1. In Bluewave, devices upload their context to a trusted cloud location (top) and modify their Bluetooth name to contain a URL (middle). Other devices can then scan for this link and download/use the information (bottom).

One way to eliminate the need for setup is to allow devices to openly broadcast some context to their neighbors. To explore this idea, we have created Bluewave, a new technique that uses Bluetooth device names to opportunistically share context. With Bluewave, devices upload context to a trusted cloud location and modify their Bluetooth name to contain a URL to this information. Other devices can then obtain this link *via* Bluetooth discovery, and download the information without having to pair or know of each other in advance (Figure 1).

Bluewave offers three contributions over conventional context-sharing techniques. First, our system provides a truly opportunistic way to share context. Rather than require devices to always be on the same network [5,6,8] or directly connect to one another [13,19,34] before they can share context, Bluewave uses a device's Bluetooth name as an out-of-band communications medium. This allows devices to reliably advertise and share context regardless as to whether or not they are on the same network. Furthermore, since all of the information needed to retrieve context is included in the device's Bluetooth name, *no pairing is required*. This allows Bluewave to be useful even when the devices sharing context have never met, making it a viable solution when devices need to share information once or spontaneously.

Secondly, Bluewave improves upon existing context-sharing frameworks in that it supports a wider range of applications. While many context-sharing systems require users to scan for beacons or devices [42], Bluewave can also let users continually broadcast their context to their environment using their existing mobile devices. This enables applications where the environment needs information about the user, and shifts the energy intensive task of scanning from a user's device to the local infrastructure. To date, the idea of openly sharing user context has been avoided due to privacy concerns [43]. To address this, we conducted an exploratory study with 15 participants, and found that users would be willing to share *some* context so long as they have explicit control over what context is shared and with whom. Informed by their feedback, we have incorporated privacy controls into Bluewave that lets users share context openly and on a per application basis. This can increase the types of context that users are willing to share through our system, which in turn increases the range of applications we can support.

Finally, Bluewave makes opportunistic context sharing practical from a developer standpoint. Our middleware makes it easy for developers to share any JSON encoded context through their applications. Additionally, we also provide real time tools to let developers see 1) what context is being shared in Bluewave, and 2) how it is specified. This allows developers to understand and utilize commonly shared contexts in their applications, as well as support both existing [25,32,36] and evolving ontologies [44].

In the next section, we provide an overview of related work, and describe the limitations of current context-sharing systems that motivated us to develop Bluewave. We then present Bluewave's architecture, and how our system uploads, share, and retrieve context. Through nine examples, we show how developers can use Bluewave to realize a wide range of context-aware applications. Finally, we provide experimental data to show that Bluewave is power efficient and has low latency, and discuss how our system's use of existing technologies make it preferable to other approaches.

RELATED WORK

The idea of having devices share contextual information has been explored in numerous works. The Active Badge [37], ActiveMap [23], and In-Out Board [27] systems, for example, allowed office workers to share information about their workplace location with their co-workers. This increased their situational awareness, and allowed them to identify new opportunities for social interaction. Hubbub [14], ConChat [26], and Community Bar [33] built on these works by also sharing information about individuals' activity and availability. This allowed users to more accurately gauge when another user is busy, thereby allowing them to determine the best time to initiate a conversation. In the Smart Party system, users shared music libraries and preferences with a centralized jukebox application [9]. The system then used this information to develop a playlist that accounted for individual users' tastes. In [20], researchers developed a mobile application to allow first responders to share location and tasks with each other during crisis scenarios. This improved responders' situational awareness while minimizing cognitive load. Finally, systems such as ErdOS [34], CoMon [19], and E2A2 [13] allow smartphones to share sensor readings when they are collocated. This allows apps to conserve power without degrading performance.

While the above work highlights individual use cases where context sharing is beneficial, their lack of generalizability has led to the creation of several middleware solutions. The GISS [10], MobilisGroups [21], Panoply [8], and CAEG [35] frameworks are designed to support context sharing for group collaboration applications. In addition to providing ontologies for sharing context, these systems also *fuse* context from individual group members, thereby increasing users' awareness of each others' activities. Meanwhile, the Solar [18], SenseWeb [16], and Mobile Gaia [6] platforms are more hardware-focused, and provide a common framework for accessing shared environmental sensors over a single environment or geographic area. Here, each device reports its sensing capabilities to a centralized coordinator (*e.g.,* a server). Applications can then query the system for context, and subscribe to the corresponding data stream.

All of the above work assumes that the devices sharing context are already connected to a common communications channel. In most real world encounters, however, connectivity is a nontrivial problem. While most mobile devices have some form of Internet access, there is no guarantee that they can send data to each other when are physically nearby. Bluetooth and Wi-Fi Direct provide this capability to some extent, but either 1) require users to manually pair devices before they can share data, or 2) rely on devices having a common shared password, which can make them susceptible to hijacking attacks [29]. Meanwhile, communication protocols such as Haggle [28], Hordes [30], and SPIN [17] allow devices to form *ad hoc* networks. However, these systems require users to install a non-standard network stack on their phones.

Our work is motivated by the realization that devices do not always have to be directly connected to share context. Instead, we programmatically manipulate a device's Bluetooth friendly name so that it contains a URL, and allow devices to download information from this URL using their own Internet connection. The idea of using radio identifiers to convey information has been explored in other works. In [7], Davies *et al.* modified the Bluetooth name of smartphones in order to contain remote control commands for public displays. A similar technique was used by Al-Akkad *et al.* to allow users to send emergency messages to first responders [1]. In that system, each phone became a Wi-Fi hotspot, and the message was encoded in the SSID.

While technologically similar, our work differs in three important ways. First, Bluewave increases the *amount* of information that can be shared through radio IDs. While prior work was limited by the size of Bluetooth names and Wi-Fi SSID fields (which are limited to 248 and 32 Bytes, respectively [3,45]), our system's use of URLs allows it to store and share any amount of context (sensor data, shopping lists, language preferences, *etc.*) at once. Secondly, while prior techniques were not designed with user privacy in mind, Bluewave incorporates controls to allow users to specify which applications are allowed to view their data. This lets authorized applications access the information they need, while preventing others from being able to anonymously "sniff" context from users as they pass by. Third, while other works focus on *short-term*, directed exchanges of information (*i.e.* sending a command or short message), our work explores the idea of manipulating Bluetooth names for *long-term* context sharing. In doing so, we aim to increase users' access to context-aware services, especially in cases where the user is new or passing through an environment, and is not aware of which services exists.

Finally, beacon technologies such as Apple's iBeacon [42] and Google's Physical Web [43] are perhaps the closest to Bluewave in terms of functionality. In these systems, dedicated hardware devices (placed in the environment) embed either a unique identifier and/or a URL in Bluetooth LE's discovery frames. Mobile devices can then scan for these frames and convert them into contextually relevant information or services using a dedicated app. Although beacons provide a fast and power-efficient way to share context, they are designed to broadcast information about an environment. In contrast, Bluewave allows users to broadcast information about themselves using their existing devices. This allows our system to support applications where the environment needs to know information about the user, thereby expanding the range of context-aware applications that can be easily created.

SYSTEM DESIGN

Given the limitations of current techniques, we set out to create a new way of sharing context that does not rely on devices explicitly pairing with one another or knowing of each other in advance. Bluewave satisfies these requirements by allowing devices to share context through their Bluetooth friendly names. In our system, each device uploads its context to a trusted cloud repository. A URL to this information is then inserted into the device's Bluetooth name and shared with all devices within range (15-30 feet).

Although conceptually simple, a number of technical challenges had to be overcome to make Bluewave possible and easy to use. In the following sections, we introduce our system's architecture. Afterwards, we present findings from an exploratory user study, and show how users' concerns about openly sharing context led us to incorporate privacy controls into our system. Finally, we describe Bluewave's developer tools, and show how developers can incorporate our system into their applications.

System Architecture

Bluewave is divided into two main components (Figure 2). The *client service* runs on individual devices, and is responsible for uploading context to the cloud and discovering nearby devices. Meanwhile, *context brokers* are hosted on web servers, and are responsible for storing and sharing context with nearby devices.

Client Service

The client service is responsible for managing and obtaining context. Each Bluewave device has exactly one client service (running in the background), and its functionality is shared between applications at runtime. The client service performs three tasks:

Task 1: Upload Context. The client service's primary responsibility is to upload context to the cloud. Each Bluewave device maintains a JSON file that represents the user/device's entire contextual state. When applications want to modify this file, they send a request to the client service. The service then makes the requested context updates and publishes the latest version to the context broker. For added efficiency, the client service only sends context *changes* to the broker. This reduces upload times, and makes uploading large amounts of context practical.

Task 2: Advertise Context. The client service's second responsibility is to advertise a device's context to its immediate neighbors. Each time the service uploads a new set of context, it modifies the device's Bluetooth name to include a URL. Other devices can then obtain this information *via* Bluetooth discovery, and download the updated context using standard HTTP (or HTTPS) requests. A sample Bluewave name is provided in Figure 3. It contains four fields, separated by colon delimiters ("::"):

- *Protocol Flag*. All Bluewave device names start with a standardized string ("BLU"). This is used to determine if a newly discovered device is Bluewave compatible.
- *Device ID*. The second field contains a device specific identifier. By default, this field contains a device's Android ID. However, any universally unique value (*e.g.,* user specified names) can be used.

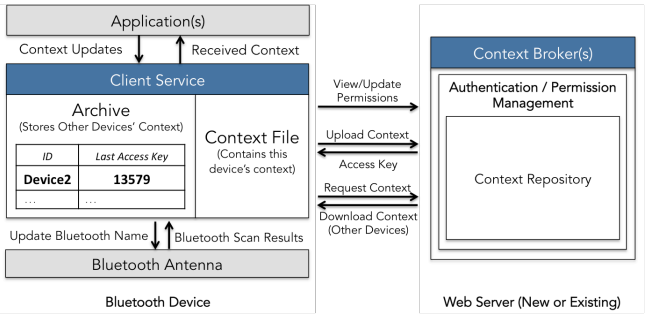


Figure 2. Bluewave's High Level Architecture

- *Context URL*. The third field contains a URL to the device's context broker. This is the link that other devices will use to obtain this device's context.
- *Download Key*. The last field contains a set of temporary credentials. When other devices want to download context, they include the download key in their request. The broker then validates the key before returning data. This key is changed after every context update to prevent devices from being able to access each other's context once they move out of range.

Task 3: Request and Obtain Context. The final responsibility of the client service is to obtain context from other devices. When directed to by an application, the client service performs periodic Bluetooth scans. As Bluewave devices are found, the service extracts the URL and download key, contacts the appropriate context broker, and forwards the information to the application for processing.

There are two technical hurdles to collecting context. The first is managing repeat discoveries. Due to the nature of Bluetooth discovery, client devices will repeatedly detect the same devices if they perform consecutive scans. To prevent devices from downloading the same context twice, each device maintains an archive of devices that it has encountered in the past. Whenever a Bluewave ID is detected, the service checks the download key to see if it has been altered. This allows the service to only download context when a new version has been posted.

A second challenge in downloading context is finding a way to minimize the amount of extraneous data that is downloaded. Many context-aware applications only need a small amount of context in order function properly. A public display, for example, may need to know the user's language preferences in order to make sure that its contents are properly translated. Similarly, a context-aware shopping agent only needs access to a user's shopping list in order to tell her what aisles to visit. To prevent devices from downloading a device's *entire* context, Bluewave requires applications to specify the JSON elements that they need in their HTTP/S request (Figure 4c). The client service will then only receive these elements from the broker.

By packaging all three capabilities into a single module, the client service provides a simple interface to Bluewave.

Figure 3. A sample Bluewave name, containing a fixed flag (A), the device's unique identifier (B), a URL to the device's context broker (C), and a temporary download key (D).

Since the client service is always running, developers can use it to share context, regardless if their application is running. More importantly, because the client service's functions are modularized, developers can turn on/off features as needed. For example, they could share context, but not scan for it. This minimizes our system's overhead.

Context Brokers

Context brokers are dedicated cloud servers that are responsible for storing and retrieving context. They serve as trusted middlemen, and allow devices to share context without having to directly connect with each other.

Context brokers have two main responsibilities. First, they provide a standardized interface for uploading and downloading context. Each context broker contains a set of PHP files that allow client services to invoke its services. To upload context, for example, the client service performs an HTTP post to *uploadContext.php*. Similarly, to download context, the client service performs an HTTP request to *getContext.php* (Figure 4). By leveraging standard web technologies, our design allows existing servers to act as context brokers. This allows Bluewave to run on existing hardware, and allows users to use their own context brokers if they choose.

Secondly, context brokers provide authentication services. When client services connect to the broker for the first time, they receive a set of authentication credentials. The client service must then provide these credentials back to the broker each time it uploads new context. Combined with secure transmission protocols such as HTTPS, this approach prevents devices from being able to modify each other's context at will. This allows devices to be confident that the context they download has not been tampered with.

While Bluewave relies on context brokers to share context, it is important to note that our architecture is designed to be decentralized. Our system neither assumes nor requires that devices utilize the same broker in order to share context. Instead, by including the URL to their broker in the Bluetooth name, Bluewave supports both centralized and decentralized configurations. This makes our system scalable, and allows it to work in any network topology.

Supporting User Privacy

Openly sharing context has significant implications from a user privacy standpoint. While our work assumes that users would be willing to share *some* context (*e.g.,* shopping lists, language preferences) with their environment, prior research has shown that preserving user privacy is an important consideration when designing Ubicomp systems

Figure 4. A sample HTTP context request, containing the URL of the device's context broker (A), its unique ID (B), the context(s) requested (C), the device's download key (D), and the unique identifier of the app requesting this information (E).

[18]. At the same time, however, prior work has also shown that users are willing to share information with nearby users/devices [40]. Consequently, we were curious as to 1) how users would react to the idea of sharing context, and 2) what types of safeguards needed to be built into Bluewave to make it socially acceptable while still being useful.

To explore these issues, we conducted a user study with 15 participants (8 males, 7 females; 21-73 years old). For this study, we created three probes (described below) which used a version of Bluewave that lacked privacy controls. We then had participants try these applications, and conducted a series of semi-structured interviews in order to elicit feedback and rate their overall comfort levels.

Preliminary Comfort with Sharing Context

To establish a baseline before the probes were introduced, participants first filled out a survey that asked them to rate how comfortable they are with openly sharing various types of context through their mobile device. This survey consisted of 12 items, 8 of which are widely accepted as being personally identifiable [22], and 4 of our own design. Each item was rated using a 5-point Likert scale (1="very uncomfortable"; 5="very comfortable"). Additionally, participants were instructed to consider each type of information separately, and to assume that sharing is limited to nearby users/devices.

The results of this survey are reported in Figure 5. Similar to prior work [40], our results show that there are a number of contexts that users are unwilling to share over any distance (*e.g.,* social security numbers, home addresses). Interestingly, though, our results also show that there *are* many types of information, including personally identifiable information that users were willing to share openly over a 30-foot radius. Many participants did not mind sharing information about their gender, age or location, because they reasoned that this information could be determined just by looking at them. Other types of information, such as shopping lists, exercise habits, and dietary restrictions, were considered more personal, but okay to share since they could not be used to identify participants in a crowd. Finally, information such as email addresses, telephone numbers, and navigation destinations were more ambiguous, with only half of the participants willing to share them openly.

Overall, these results support the idea of using a system such as Bluewave. While our participants were unwilling to share all 12 types of information, they *were* open to the idea of openly sharing a subset over a short distance. We are aware that our results cannot be used to definitively say what contexts can and cannot be shared opportunistically. However, our results show that this distinction exists, and

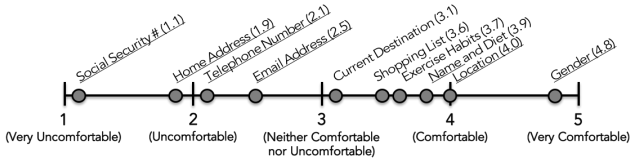


Figure 5. Participants' initial comfort levels with sharing various types of contextual information. Underlined items indicate personally identifiable information.

that there is a wider range of information that users might be willing to share than what one might assume.

Probing Applications

After the survey, we introduced participants to our probes. The criteria we used to select the three probes were they had to be typical Ubicomp applications that others had built, and that using standard development approaches, required significant *a priori* configuration in order to work.

For the first probe, we created a series of intelligent signs for our institution's library. Using a custom Android application (Figure 6, top left), participants could search through the library catalog and select a book to check out. The title and call number of the book was then shared with the signs *via* Bluewave, allowing them to either guide the user to the correct row/column, or inform the user that the book is not available (Figure 6, top right).

The second probe was a "virtual concierge" for public spaces. For this example, we deployed a series of Bluewave equipped sensors at the entrances of our institution's University Center. By having users share their email address, dietary restrictions (*e.g.,* lactose intolerant), and exercise habits through Bluewave, our system could determine 1) if the user has been to the building before, 2) which restaurants he/she can order food from, and 3) what exercise facilities he/she might be willing to use. The system could then send a notification to users' phones, which welcomed them, and provided a list of services that they might find interesting (Figure 6, middle).

Finally, we created a shopping assistant to help users find items in our institution's bookstore. Here, participants used a custom app to share their shopping list *via* Bluewave. As participants entered the bookstore, specially placed sensors scanned the user's shopping list and determined which items were available for purchase. This information was then presented to the user (Figure 6, bottom). Additionally, our shopping assistant also uses the context shared by our library application to see if the title they were looking for is offered in the store. This provided users with an alternative way to acquire a book of interest—one they might not have considered when they first entered the store.

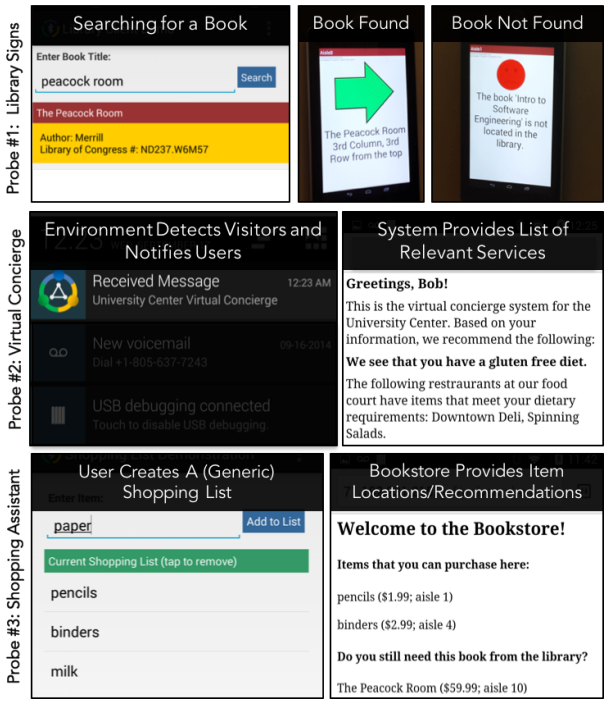


Figure 6. Probes developed for our user study.

Post Study Comfort Levels

After using our probes, we let participants adjust their self-reported comfort levels from the first part of the study to see if our probes had any influence on their willingness to share context. In *all 15 cases*, we found that participants' willingness to share information increased. While participants' opinions did not change for items that they had ranked high or low, they did increase their scores for middle ranked items (rating 2-4) by one point, with the most popular being shopping lists (9 out of 15 participants), exercise habits (6 out of 15), and diet (5 out of 15).

In follow-up interviews, we asked participants to explain the rationale behind their increased scores. In each case, participants told us that they were initially hesitant to share their context ("I don't want to give others [my information] and get nothing in return" (P9)). Yet after seeing our prototypes, many stated that they had a better idea of how sharing context might benefit them. This made them more willing to have this information be openly available.

Adding Privacy-Centric Features

While the participants in our study warmed up to the idea of openly sharing context, the issue of user privacy was brought up on numerous occasions. Many participants liked the services provided by our probes, but were uncomfortable with the idea that their information could be *potentially* viewed by anybody. Furthermore, while participants admitted that most of the information used by our probes was innocuous (*e.g.,* "I eat junk food—anybody can know that" (P13)), they also noted that they would feel more comfortable if they had explicit control over what information is being shared: "I would prefer the system to ask me before taking my information." (P6).

In light of these findings, we have incorporated several privacy features to Bluewave. Our revised system now requires developers to register their applications on our framework's website and receive an app ID. This ID must be provided to context brokers to receive context (Figure 4e), thereby preventing nonregistered applications from being able to collect context anonymously. Additionally, context brokers are now able to track individual applications in order to see what types of information they ask for. Users can then receive a notification on their devices (*via* the client service) when applications request context for the first time (Figure 7a), and be directed to a web interface to grant or deny access (Figure 7b). Finally, in light of our discovery that there are *some* contexts that users are comfortable sharing at all times, we now allow users to specify which contexts (*e.g.,* location, email addresses) they would like to share openly using Bluewave (Figure 7c). This prevents users from having to approve *every* request for context, while still allowing them to be notified in outstanding cases.

Although these features give users precise control over how their context is shared, it requires users to grant applications permission before they can receive context-aware services. While this limits their ability to opportunistically receive services, our study shows that such safeguards are needed to give users peace of mind. Thus, while Bluewave still allows users to openly share context if they choose, our design recognizes that the decision must be left to them.

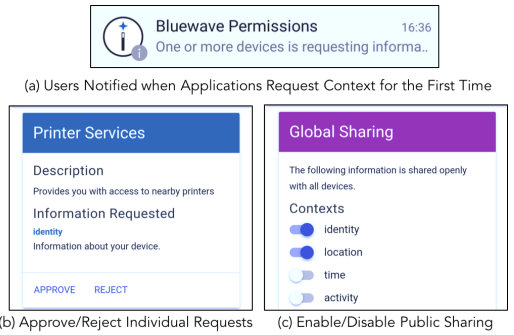
Supporting Developers

While Bluewave was designed to address many usability issues of context-aware applications, we also wanted to make it easier for *developers* to leverage opportunistic context sharing in their applications. The first way we support this goal is by making Bluewave *programmatically* easy to use. We have created a lightweight Android library that automates our system's core processes (*e.g.,* uploading and/or downloading context). Developers can then easily utilize Bluewave's functionality by adding this library to their code and performing the following steps:

Step 1: Create the Client Service. The first step in using Bluewave is to instantiate the client service that loads the device's context file into memory. Developers can then add, remove, and/or modify context as needed.

Step 2: Specify Context(s) to Receive. To receive context, developers must 1) register their application with our website and 2) provide their app ID, as well as the specific JSON elements they are looking for, to the client service. The service will then automatically scan for devices and download the requested information.

Step 3: Process Incoming Context. Each time the client service downloads a new or updated context file, it forwards this information (along with standard Bluetooth



(a) Users Notified when Applications Request Context for the First Time

(b) Approve/Reject Individual Requests (c) Enable/Disable Public Sharing

Figure 7. Bluewave notifies users when applications request context for the first time (a). Users can then use our interface to share context with individual apps (b) or openly (c).

discovery data, such as the MAC address and RSSI value) to a predefined event handler. Developers can then use this information as needed by their particular application.

The second way that Bluewave supports developers is by allowing them to see how context is being shared across our system. Each time a context broker sees a new type of context, it uploads a sample of that information to our centralized database. This information is then used to create a web dashboard that documents 1) which contexts are available, 2) how they are defined, and 3) how often they are openly shared (Figure 8). Through our dashboard, developers can see which contexts are most commonly used. This allows developers to utilize the same contexts in their application without having to explicitly coordinate.

The inclusion of "living documentation" gives Bluewave a significant advantage over other context-sharing systems. To date, other systems have assumed that developers will share context using a standard ontology [36]. In contrast, Bluewave can share *any* JSON encoded context, making it compatible with a wide range of existing, as well as emerging, standards [44,46]. In the following sections, we present nine applications built with Bluewave. We use them to validate the utility of our tools, and show how our system offers a simple but effective way to share context.

VALIDATION

We validate Bluewave in two parts. In the next section, we show how the ability to openly share context is useful in a wide range of applications. Afterwards, we evaluate Bluewave's battery consumption and latency, and show that our system is practical along both dimensions.

Example Applications

To demonstrate Bluewave's usefulness as a *technology*, we have created a total of nine applications using our system (Figures 9-10). We chose applications from two domains, public displays and the Internet of Things (the first of which has been of long-standing interest to Ubicomp research and the second of which has recently attracted a lot of attention in industry) to show how Bluewave increases the types of context-aware services that can be easily created.

Additionally, our applications also showcase Bluewave's ability to reuse context. Rather than create all nine

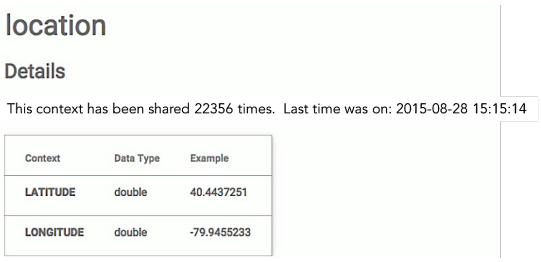


Figure 8. Bluewave's dashboard is automatically populated by context brokers, and allows developers to see which contexts are being shared, and how they are formatted.

examples ourselves, we gave Bluewave (and our dashboard) to three different developers, and asked them to create applications using our system. In doing so, we not only show that Bluewave is easy to understand and use, but also that our system makes it practical for developers to utilize the same context(s) in different applications.

Public Displays

A key challenge in creating public displays is scalability. While prior research has shown that we can intelligently modify the contents of signs based on users' context, the resulting systems require users to manually connect to the sign [2,24], or register with a central service [4,11,12]. This works when the number of public displays is small, but becomes impractical if we want to expand this functionality beyond a single environment or to a large number of users.

Bluewave provides a simple but elegant solution to this problem. Instead of requiring users to explicitly connect to every sign, our system allows signs to scan and collect context from users. The signs can then adjust their contents based on users' collective information needs. To demonstrate this, one of our developers created a series of self-translating signs using our system (Figure 9a-b). For this demonstration, users share their operating system language settings *via* Bluewave. The signs then detect the language and translate their contents to accommodate the most common language(s) for the users nearby. In later work, the same developer augmented the signs to support visually impaired users. By having users share their disability *via* Bluewave and using Bluetooth RSSI to estimate range, the signs are able to detect when a user with special needs is nearby. The signs then assist these users by speaking out loud (Figure 9c).

We have also been using Bluewave to create navigation aids. For this example, we created a map application (Figure 9d) that shares users' navigation destination (latitude/longitude coordinates) through Bluewave. A developer then created a smart bus stop sign that can listen for this information, and tell the user which bus routes will travel near this location (either by drawing a red box around the route name, or displaying a user specified icon; Figure 9e). Another example, created by a different developer, uses *the same context* to control a house's porch lights (Figure 9f-g). Here, a Bluewave-equipped sensor placed at a house searches for users that are navigating towards it.

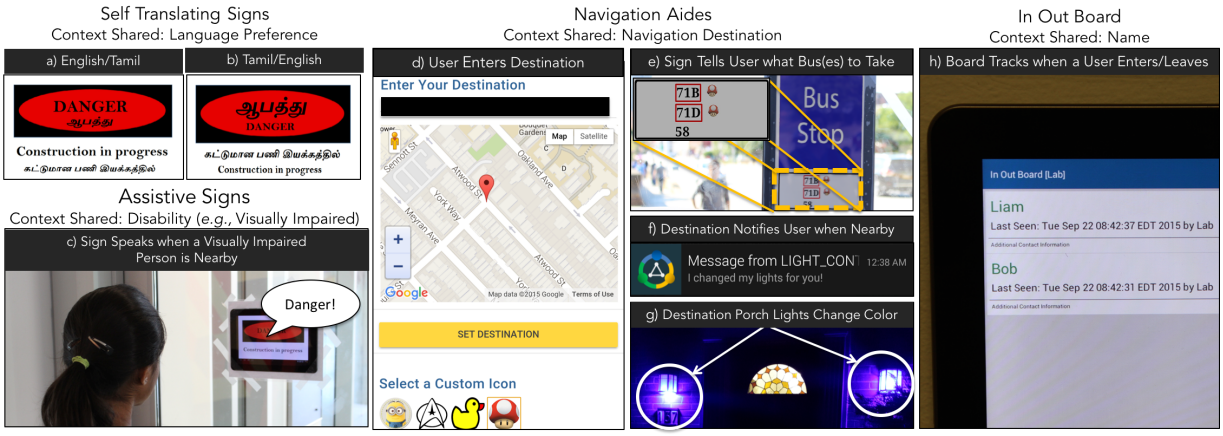


Figure 9. Public displays created with Bluewave. The contexts needed for these applications are noted above.

When found, the sensors alter the color of the porch lights and sends the user a message (*e.g.,* "Look for the purple lights"), thereby showing how the same context can be used in more than one way.

We also used Bluewave to recreate classic context-aware signs, such as the In-Out Board [27]. Our "sign" is a modified tablet that has been placed at the entrance of our lab (Figure 9h). To use our system, users share their name *via* Bluewave. The sign can then display this name, as well as the time they were detected so that other users can know who is in the office. Unlike the original In-Out Board, which requires users to place a dedicated hardware device (*i.e.,* an "iButton") on the sign to check in/out, our system uses Bluewave to continually determine who is nearby. This eliminates the need for users to directly interact with our sign, and allows it to seamlessly work with visitors and changing lab membership—a feature that the original system was not designed to handle.

In each case, Bluewave improves upon the state-of-the-art in that public displays can now provide users with customized services without having to know of them in advance. Clearly, we kept these prototypes simple to illustrate the main benefits of using Bluewave and did not address usability issues such as how to create a self-translating sign that can handle large numbers of users. The main goal with our examples was to show how sharing a small amount of information *via* Bluewave substantially increases users' access to timely and relevant information.

Internet of Things

We are also exploring how Bluewave can be used to support the Internet of Things (IoT). As an initial demonstration, we have created a series of remote sensors (*i.e.,* tablets/phones) that can monitor and share their ambient temperature, barometric pressure, and humidity levels using Bluewave. Users (using a scanning app created by another developer) can then collect and view this information from their mobile phones without having to directly connect with each sensor (Figure 10a). Due to Bluetooth's latency, our early prototypes focused on sensors whose readings do not quickly change. Since then, however, our system now shares TCP connection settings (*e.g.,* IP address/port) in addition to sensor data. This allows devices to connect to individual sensors, and shows how we can use Bluewave to support occasions where real-time context (*e.g.,* microphone data) is required.

Another use for Bluewave in IOT is to transfer user preferences between smart environments (Figure 10b-c). Inspired by systems such as Aura [31], which first introduced the idea of cross-environment syncing, we created a mobile app that allows users to adjust the color and brightness of Wi-Fi connected light bulbs in their room. These settings are then shared *via* Bluewave, and used to configure the lights when the user enters a new location (*e.g.,* an office and/or hotel). While this prototype focuses on light settings, we are also looking at how the same technique can be used to migrate other settings (*e.g.,* preferred temperature). This would allow environments to intelligently configure themselves without requiring the user to perform manual configuration.

A third use for Bluewave in IoT is to support *ad hoc* facial recognition (Figure 10d). For this example, one of our developers created an app that lets users upload photos of themselves to a web server. The URLs to these photos, along with the user's name (using the same format as the one used by our In-Out Board example), is shared *via* Bluewave. Camera-equipped devices can then obtain this context, and use it to identify specific users in the environment. This allows devices to tell if a user is looking at them (rather than assume that this is the case when they are in Bluetooth range) without requiring 1) developers to create a centralized face database, or 2) users to manually register their face with every environment.

Our fourth, and most ambitious use of Bluewave to date is to make it easier to create and deploy IoT applications. As part of our ongoing work, we have created a series of smart appliances that can share their context (*e.g.,* their type and capabilities) using Bluewave. We then created a custom Python environment that can scan for nearby devices, and convert them into in-code objects. Developers can then use these objects to control the appliances programmatically.

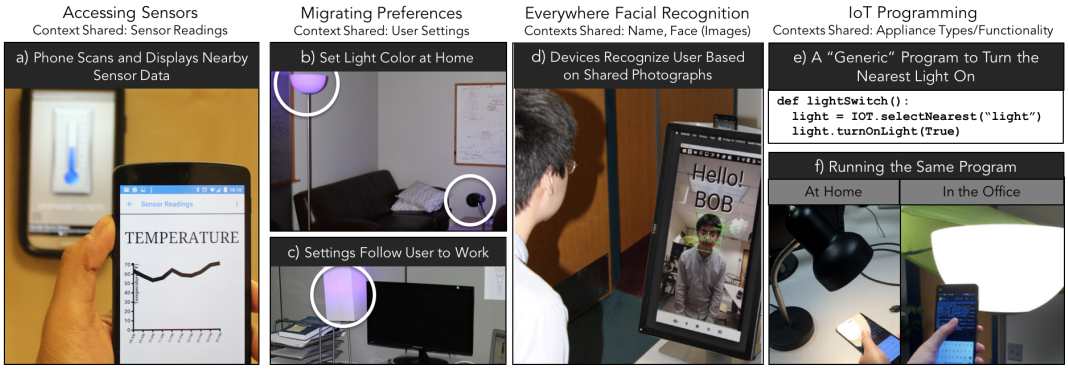


Figure 10. Internet of Things applications created using Bluewave

Configuration	Avg Battery Drain
Control (Bluetooth Not Discoverable)	3.2%
Configuration 1: (Discoverable Only)	4.6%
Configuration 2: (Discoverable/Scanning)	12%

Table 1. Bluewave battery drain after 8 hours

Technique	Avg Download Time
Bluewave	11.6s
Bluetooth Direct Connection	55.8s
Physical Web + Download (BLE)	9.3s

Table 2. Speed test results (averaged over 30 trials)

Additionally, our environment also lets developers refer to appliances by their generic type (*e.g.,* "lights"; Figure 10e). Our system then uses Bluewave to dynamically determine which appliance(s) of this type are nearby at runtime, thereby allowing the same code (*e.g.,* "turn on the nearest light") to be executed in different locations (Figure 10f).

Collectively, these examples demonstrate how the ability to openly share context is relevant from an IoT perspective. To date, research in IoT has focused on the challenges of instrumenting an individual environment, such as a home or office. Our work complements this growing body of research by showing how we can create services that can span these smart environments. This increases users' access to services while minimizing the need for setup.

Battery Consumption

To evaluate Bluewave's power requirements, we conducted a series of battery drain tests using identically configured Samsung Galaxy S IV smartphones. First, we measured the power drain on each device when idle (*i.e.,* no applications or Bluetooth running). We then tested two configurations of Bluewave for eight hours (each test was run 5 times):

- **Configuration 1:** The device is only sharing context (*i.e.,* Bluetooth name is continuously discoverable).
- **Configuration 2:** The device is both sharing context and listening for context from others (*i.e.,* Bluetooth discovery is set on a continuous loop, and Bluetooth name is set to be continuously discoverable).

As expected, using Bluewave results in increased battery consumption (Table 1). However, the exact amount varies depending on how it is used. When Bluewave is configured to share context (Configuration 1), our system only consumed an additional 1.4% of battery power after eight hours. When the device is both sharing and listening for context (Configuration 2), Bluewave's energy usage was almost four times as much as the baseline condition, but still relatively low at 12%.

These results are significant for two reasons. First, they show that it is feasible to use Bluewave for extended periods of time. While we do not expect user devices to have to constantly scan their environments *and* share context, our results show that their devices can do so for a long period of time without depleting their battery.

Our results also show that, in many use cases, Bluewave's impact on battery life is minimal. Of the 12 prototypes presented in this paper, only two require the user's device to scan the environment; the rest allow users to simply broadcast their context and for the environment to scan. As Bluewave is intended primarily to allow users to share context with the environment, we expect that users will only need to have their Bluetooth set to discoverable in order to benefit from our system. This allows Bluewave to run in its low configuration for most use cases, while still allowing users to scan for context if and when needed.

Latency

To evaluate Bluewave's latency, we ran a series of time trials. For these trials, we positioned 10 Bluewave devices throughout an office area (to simulate the room when full), and timed how long it took for an 11th device (a Galaxy S IV) to scan and download their context. We then compared this to the time needed to download the same information by 1) connecting to each device *via* Bluetooth (assuming no PIN is required), and 2) using Physical Web beacons (*i.e.,* using Bluetooth LE (BLE) to discover each beacon, extract the URL, and download the file). Each method was tested 60 times, and the best 30 times are reported in Table 2.

Our results show that Bluewave's latency outperforms, or is at least comparable to, industry standard techniques. Our system is able to discover and obtain context nearly *five times faster* than using Bluetooth pairing. This is because Bluewave is able to discover devices and download context in parallel, while Bluetooth must perform each step in sequence. Our system is slower than using Physical Web beacons to share context (Table 2, row 3), due to

Bluetooth's longer discovery time. Our results, however, show that the difference is, on average, 2.3s. Practically speaking, this means that any application that can tolerate Physical Web's latency should also work with Bluewave.

These results show that Bluewave offers a good compromise between speed and flexibility. While our system is not the fastest, it offers similar performance to state-of-the-art technologies. This, combined with our system's reliance on standard Bluetooth, makes our system useful in a wide range of situations today, without requiring users to have to purchase additional hardware.

DISCUSSION

In this section, we examine our prototypes' reliance on specialized applications to share context, and show how this dependency should decrease as our system becomes more widely available. Afterwards, we rationalize our decision to base Bluewave on Bluetooth as opposed to BLE, and highlight the need to eventually support both technologies.

The Need for Context Sharing Applications

Although Bluewave provides a novel way to share context, our current prototypes rely heavily on custom apps to publish and share user information. For example, before users can receive recommendations from our intelligent bus displays, they need to specify their destination using our map application. Similarly, before our facial recognition system can identify a person, that user must first use our app to collect and share his/her photograph and name.

While having dedicated apps for every system goes against the idea of "opportunistic" context sharing, it is important to note that the apps used in our examples are not system specific. Our map application, for instance, does not just share the user's destination with bus signs, but rather with any system that needs this information (including our porch light system, as mentioned in the previous section). Likewise, our facial recognition app does not just share user photographs with a single environment, but with any environment that needs it. This allows devices to recognize users regardless if they have been there before.

For now, we created our own context-sharing applications in order to show how our system might one day be used. In the future, however, many of the contexts used by our example applications can be obtained directly from the operating system, or bootstrapped into "standard" apps (*e.g.,* a map or shopping list app). This would reduce the need for our specialized apps, while still allowing context-aware systems to get the information they need.

Bluetooth *vs.* Bluetooth Low Energy

Bluewave currently relies on classic Bluetooth in order to share context. However, Bluetooth 4.0 (*i.e.,* Bluetooth Low Energy) has been ratified since 2010, and is available on commercial hardware. While our experiments show that it is practical to have devices scan and be discoverable using Bluetooth for a single day, BLE devices are more efficient, and can broadcast for *years* on a single charge [15].

Our decision to base Bluewave on Bluetooth was made for compatibility reasons. Despite being available for nearly 5 years, only 60% of Android devices have the Android version (4.3+) needed to *scan* for BLE devices [47] and, of the thousands of Android models currently on the market, only the Nexus 6 and 9 (to our knowledge) have the ability to serve as BLE beacons [48]. In contrast, classic Bluetooth is available on nearly every Android device, and allows devices to both scan for devices and be discoverable at the same time. We have successfully tested *and* deployed Bluewave on devices running Android Gingerbread, a 5-year-old operating system. As a result, while Bluetooth is not as efficient as BLE, its ability to run on virtually any hardware, combined with its low energy use and latency, makes it the preferred solution for now.

Nevertheless, we acknowledge that BLE's benefits are too great to ignore. Work done by Google's Physical Web shows that it is possible to share URLs *via* BLE beacons. As a result, when more devices support BLE beaconing, we will extend Bluewave to support this technology.

CONCLUSION

This paper contributes Bluewave, a new technique that allows devices to share contextual information through their Bluetooth names. We described the technical details of our system, and showed how our system can be used to reliably share context with nearby devices, regardless of whether they have ever met before. We also conducted a user study with 15 participants, and showed how their initial reluctance to share (some) information openly has caused us to incorporate privacy features. Through nine prototypes, we show how developers can easily create applications that use both new and commonly shared contexts. Finally, we evaluated Bluewave's energy consumption and latency, and found that it is feasible along both dimensions.

Bluewave takes an important step towards making opportunistic context-sharing practical outside of the lab. However, there are still aspects of our system that warrant further study. Currently, our system relies on cloud servers to share context. In future releases, we are interested in augmenting the client service so that the context broker can be hosted on user devices. This would allow our system to work in environments without Internet connectivity.

Additionally, while our probes show that users recognize the value of being able to share context, it is important to note that these findings only represent their initial impressions. In future work, we will release Bluewave to the developer community, and examine 1) the types of applications created, and 2) how users utilize it in their everyday lives once the novelty period wears off. This knowledge will provide us with a more comprehensive look at how Bluewave is being used, and will help reveal which use cases are worth studying in greater detail.

REFERENCES

1. Amro Al-Akkad, Leonardo Ramirez, Alexander Boden, Dave Randall, and Andreas Zimmermann. 2014. Help Beacons: Design and Evaluation of an Ad-Hoc Lightweight SOS System for Smartphones. *CHI '14*, 1485–1494. http://doi.org/10.1145/2556288.2557002

2. Rafael Ballagas, Michael Rohs, and Jennifer G Sheridan. 2005. Sweep and point and shoot: phonecam-based interactions for large public displays. *CHI '05 Extended Abstracts*, 1200–1203.

3. S I G Bluetooth. 2013. Bluetooth specification version 4.1. *Bluetooth SIG*.

4. Scott Carter, Elizabeth Churchill, Laurent Denoue, Jonathan Helfman, and Les Nelson. 2004. Digital Graffiti: Public Annotation of Multimedia Content. *CHI '04 Extended Abstracts*, ACM, 1207–1210. http://doi.org/10.1145/985921.986025

5. Guanling Chen, Ming Li, and David Kotz. 2008. Data-centric middleware for context-aware pervasive computing. *Pervasive Mobile Computing* 4, 2: 216–253. http://doi.org/10.1016/j.pmcj.2007.10.001

6. Shiva Chetan, Jalal Al-Muhtadi, Roy Campbell, and M. Dennis Mickunas. 2005. Mobile Gaia: A Middleware for Ad-Hoc Pervasive Computing. *CCNC '05*, 223–228. http://doi.org/10.1109/ccnc.2005.1405173

7. Nigel Davies, Adrian Friday, Peter Newman, Sarah Rutlidge, and Oliver Storz. 2009. Using bluetooth device names to support interaction in smart environments. *Mobisys '09*, 151–164. http://doi.org/10.1145/1555816.1555832

8. Kevin Francis Eustice. 2008. *Panoply: active middleware for managing ubiquitous computing interactions*. ProQuest.

9. Kevin Eustice, V. Ramakrishna, Nam Nguyen, and Peter Reiher. 2008. The Smart Party: A Personalized Location-Aware Multimedia Experience. *Consumer Communications and Networking Conference*, 873–877.

10. Alois Ferscha, Clemens Holzmann, and Stefan Oppl. 2004. Context Awareness for Group Interaction Support. *MobiWac '04*, 88–97. http://doi.org/10.1145/1023783.1023801

11. Saul Greenberg, Michael Boyle, and Jason Laberge. 1999. PDAs and shared public displays: Making personal information public, and public information personal. *Personal Technologies* 3, 1-2: 54–64. http://doi.org/10.1007/BF01305320

12. Elaine M Huang and Elizabeth D Mynatt. 2003. Semi-public displays for small, co-located groups. *CHI '03*, 49–56. http://doi.org/10.1145/642611.642622

13. Yun Huang, Anthony Tomasic, Yufei An, Charles Garrod, and Aaron Steinfeld. 2013. Energy Efficient and Accuracy Aware (E2A2) Location Services via Crowdsourcing. *WiMob '13*, 436–443. http://doi.org/10.1109/WiMOB.2013.6673396

14. Ellen Isaacs, Alan Walendowski, and Dipti Ranganthan. 2002. Hubbub: A Sound-Enhanced Mobile Instant Messenger that Supports Awareness and Opportunistic Interactions. *CHI '02*, 179–186. http://doi.org/10.1145/503376.503409

15. Sandeep Kamath and Joakim Lindh. 2010. Measuring bluetooth low energy power consumption. *Texas instruments application note AN092, Dallas*.

16. Aman Kansal, Suman Nath, Jie Leu, and Feng Zhao. 2007. SenseWeb: An Infrastructure for Shared Sensing. *MultiMedia, IEEE* 14, 4: 8–13. http://doi.org/10.1109/mmul.2007.82

17. Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. 2002. Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks. *Wireless Networks* 8, 2-3: 169–185. http://doi.org/10.1023/a%253a1013715909417

18. Marc Langheinrich. 2001. Privacy by Design — Principles of Privacy-Aware Ubiquitous Systems. In *Ubicomp '01*. 273–291. http://doi.org/10.1007/3-540-45427-6_23

19. Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. 2012. CoMon: Cooperative Ambience Monitoring Platform with Continuity and Benefit Awareness. *MobiSys '12*, 43–56. http://doi.org/10.1145/2307636.2307641

20. Grace Lewis, Marc Novakouski, and Enrique Sánchez. 2013. A Reference Architecture for Group-Context-Aware Mobile Applications. In *Mobile Computing, Applications, and Services*. 44–63. http://doi.org/10.1007/978-3-642-36632-1_3

21. Robert Lubke, Daniel Schuster, and Alexander Schill. 2011. MobilisGroups: Location-Based Group Formation in Mobile Social Networks. *PERCOM '11*, 502–507. http://doi.org/10.1109/percomw.2011.5766941

22. Erika McCallister, Timothy Grance, and Karen A. Scarfone. 2010. SP 800-122. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII).

23. Joseph F McCarthy and Eric S Meidel. 1999. Active Map: A Visualization Tool for Location Awareness to Support Informal Interactions. *HUC '99*, 158–170.

http://doi.org/10.1007/3-540-48157-5_16

24. Matei Negulescu and Yang Li. 2013. Open project: a lightweight framework for remote sharing of mobile applications. *UIST '13*, 281–290. http://doi.org/10.1145/2501988.2502030
25. Maria Poveda Villalon, Mari Carmen Suárez-Figueroa, Raúl García-Castro, and A. Gómez-Pérez. 2010. A Context Ontology for Mobile Environments.
26. Anand Ranganathan, Roy Campbell, Arathi Ravi, and Anupama Mahajan. 2002. ConChat: A Context-Aware Chat Program. *Pervasive Computing* 1, 3: 51–57. http://doi.org/10.1109/mprv.2002.1037722
27. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. 1999. The context toolkit. *CHI '99*, 434–441. http://doi.org/10.1145/302979.303126
28. James Scott, Jon Crowcroft, Pan Hui, and Christophe Diot. 2006. Haggle: a Networking Architecture Designed Around Mobile Users. *WONS '06*, 78–86.
29. Ajay Sharma. 2008. Bluetooth security issues: threats and consequences. *Proc. of the 2nd national conference COIT*, 78–80.
30. Clay Shields and Brian Neil Levine. 2000. A protocol for anonymous communication over the Internet. *CCS '00*, 33–42. http://doi.org/10.1145/352600.352607
31. João Pedro Sousa and David Garlan. 2002. Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Software Architecture*. Boston, MA, 29–43. http://doi.org/10.1007/978-0-387-35607-5_2
32. Xiao Hang Wang, Hung Keng Pung, Da Qing Zhang Tao Gu. An Ontology-based Context Model in Intelligent Environments. Retrieved September 17, 2015 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.2194
33. Kimberly Tee, Saul Greenberg, and Carl Gutwin. 2006. Providing Artifact Awareness to a Distributed Group Through Screen Sharing. *CSCW '06*, 99–108. http://doi.org/10.1145/1180875.1180891
34. Narseo Vallina-Rodriguez and Jon Crowcroft. 2011. ErdOS: Achieving Energy Savings in Mobile OS. *MobiArch '11*, 37–42. http://doi.org/10.1145/1999916.1999926
35. Bin Wang, J Bodily, and S K S Gupta. 2004. Supporting Persistent Social Groups in Ubiquitous Computing Environments Using Context-Aware Ephemeral Group Service. *PerCom '04*, 287–296. http://doi.org/10.1109/percom.2004.1276866
36. X.H. Wang, Da Qing Zhang, Tao Gu, and H.K. Pung. 2004. Ontology based context modeling and reasoning using OWL. *PERCOM '04*, 18–22. http://doi.org/10.1109/PERCOMW.2004.1276898
37. Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. 1992. The active badge location system. *ACM Transactions on Information Systems* 10, 1: 91–102. http://doi.org/10.1145/128756.128759
38. Mark Weiser. 1991. The Computer for the 21st Century. *Scientific American* 265, 3: 94–104. http://doi.org/10.1038/scientificamerican0991-94
39. Mark Weiser. 1993. Some computer science issues in ubiquitous computing. *Communications of the ACM* 36, 7: 75–84.
40. Jason Wiese, Patrick G Kelley, Lorrie F Cranor, Laura Dabbish, Jason I Hong, and John Zimmerman. 2011. Are You Close with Me? Are You Nearby?: Investigating Social Groups, Closeness, and Willingness to Share. *Ubicomp '11*, 197–206. http://doi.org/10.1145/2030112.2030140
41. Craig Wisneski, Hiroshi Ishii, Andrew Dahley, et al. 1998. Ambient displays: Turning architectural space into an interface between people and digital information. In *Cooperative buildings: Integrating information, organization, and architecture*. Springer, 22–32.
42. Apple - iBeacon for Developers. Retrieved September 18, 2015 from https://developer.apple.com/ibeacon/
43. The Physical Web. Retrieved March 16, 2015 from https://google.github.io/physical-web/
44. Project Brillo | Google Developers. Retrieved September 17, 2015 from https://developers.google.com/brillo/?hl=en
45. IEEE 802.11: Wireless LANs. Retrieved September 17, 2015 from http://standards.ieee.org/about/get/802/802.11.html
46. Schema.org. Retrieved September 17, 2015 from https://schema.org/
47. Dashboards | Android Developers. Retrieved September 18, 2015 from https://developer.android.com/about/dashboards/index.html
48. Building an Android Beacon (Android iBeacon Tutorial Overview) - PubNub. Retrieved September 18, 2015 from http://www.pubnub.com/blog/building-android-beacon-android-ibeacon-tutorial-overview/