Rethinking Context for Pervasive Computing: Adaptive Shared Perspectives

Christine Julien, Agoston Petz, and Evan Grim

The Center for Advanced Research in Software Engineering
The Department of Electrical and Computer Engineering
The University of Texas at Austin

Email: {c.julien, agoston, evangrim} @mail.utexas.edu

TR-ARiSE-2012-008

ARiSE
Advanced Research in Software Engineering

Rethinking Context for Pervasive Computing: Adaptive Shared Perspectives

Christine Julien, Agoston Petz, and Evan Grim

The Center for Advanced Research in Software Engineering
The Department of Electrical and Computer Engineering
The University of Texas at Austin
Email: {c.julien, agoston, evangrim}@mail.utexas.edu

***Abstract*—Mobile and pervasive computing applications depend on environmental awareness, not just of their own local system and immediate surroundings, but increasingly of the network as a whole. Traditional models of context do not sufficiently account for the trade-off between accuracy, availability, and efficiency when considering the mechanisms by which context can be gathered and shared in mobile, ad-hoc networks. Traditional approaches to context remain *egocentric*, but next generation environments demand coordination among entities and access to shared context resources, and thus context must be treated as a *combination* of local and shared information. We motivate our perspective on the future of context in mobile pervasive environments staring with historical framing of the problem, and present our recent efforts in adaptive shared context perspectives.**

I. Introduction

Many of today's computing systems are characterized by very large numbers of interconnected devices that represent users and their changing digital situations. These devices are integrated with the environment and connected to each other via highly volatile and resource constrained wireless links. Such networked environments include, among many others, pervasive computing systems, mobile networks, intelligent transportation systems, and the Internet of Things. In these environments, an entity's *situation* is characterized by myriad facets of the nearby physical and cyber environments. To fully characterize its situation, an entity must assess its local information, including sensing its own capabilities, status, location, and environmental conditions.

Traditional approaches to *context-awareness* have provided these views, but they remain largely *egocentric*. Specifically, research in context-aware computing has created egocentric applications that adapt to location (e.g., in tour guides [1]), time (e.g., in reminder applications [12]) and even weather conditions (e.g., in automated field-note taking [32]). Several toolkits have long provided abstractions for accessing such context information [13], [19], [21].

In emerging environments, entities must also coordinate with other entities via the wireless network to assess *shared* local conditions, including the state of the network, availability of data and resources, physical characteristics of the environment, and social network connections. Consider an opportunistic network of mobile devices in a public park where collective context identifies a group of people interested in a pick-up game of football and having similar skill. Alternatively, a device on an automobile may generate an individualized group containing nearby automobiles that can collide with it. Knowledge about connection qualities in a neighborhood of a dynamic mobile network can enable nodes to jointly select the best routing protocol for a region of the network [24]. For many emerging applications, the availability of the *combination* of egocentric and shared context information is essential because it enables these applications to adapt to the current situation and take advantage of currently available resources.

Generating context data is becoming increasingly easier and cheaper. Any run-of-the-mill smartphone is now shipped with myriad on-board sensors. Popular consumer devices support various personal and social sensing tasks, including many varieties of networked fitness monitors. In the Internet of Things (IoT) [3], ordinary objects become *smart objects*, imbued with computation, sensing, and networking capabilities [7], [27]. Participatory sensing [9] relies on these new capabilities and explicitly allows users to sense and share local information. The Open Data Kit (ODK) [20] connects with participatory sensing applications to aid in collecting expressive context. Extensions to ODK, e.g., ODK Sensors [8], provide programming constructs that simplify access to external sensors connected to smart devices. Dandelion [29] and Gadgeteer [39] have similar motivations. Our own work has recently focused on how to assess shared context [16], [23].

Given the significant increase in both egocentric and shared context availability, the ability to and interest in creating adaptive behaviors has also increased. In Section II, we take a retrospective look at context and context-awareness capabilities, given our changed perspective on context and the importance of context sharing. Key foci of this discussion are the dual notions of *efficiency* and *expressiveness* of context acquisition and sharing. Efficiency is extremely important, as existing context acquisition and sharing mechanisms have largely proven to have too high of an overhead to be practically feasible. Expressiveness is important in ensuring that the right context is acquired with the right level of quality (or *fidelity*) for application requirements.

We rely on this history to motivate a shift in perspective from egocentric approaches to collaborative and cooperative

approaches to context awareness. In Sections III and IV, we present some of our own recent work related to acquiring and sharing context information, both conceptually and through practical systems contributions. Given the increasing availability of expressive context information that is efficiently gathered, Section V looks at potential untapped *uses* of context, which largely entail *adaptation* of user and application behavior. We close with a forward-looking view on the application of this new view of context to pervasive computing and the potential impacts on the widely varied research landscape.

II. A History of Context

We begin our discussion with a framing of the history of context-awareness and context sensing. In the next section, we use this view to motivate our shifting perspective on the need for collaborative and cooperative sensing of context information with the potential to increase the expressiveness and efficiency of context awareness.

Context awareness is crucial in developing dynamic pervasive computing applications, and recognition of this has driven abundant research in how context can be more efficiently sensed and shared. One of the earliest contributions, the Context Toolkit [37], draws inspiration from the graphical user interface (GUI) domain and proposes abstracting context producers into "widgets" that ease the burden in developing context enabled applications. Building on this foundation the Context Fabric [21] proposed a broader vision where context needs would be met by an underlying infrastructure, and [19] further demonstrated how generalizing notions of context and simplifying the burden placed on developers serve as important goals in facilitating flexible and powerful applications.

These early contributions provide important capabilities, but the resource-constrained systems used in pervasive computing applications require extremely efficient solutions. This has guided additional efforts to focus on identifying ways to provide these capabilities with a primary goal of minimizing the processing and network resources necessary to support them. Researchers have found this efficiency using a variety of methods, such as relying on passive eavesdropping [4], [34], driving sensing at the application level [41], or by sending only changes to context values [25].

In addition to focusing on efficiency, researchers are also exploring how low-level data can be aggregated into useful higher-level context information. In [26] context is aggregated from a personal network to automatically add semantic tags to documents created or accessed on a personal device. Similarly, the Solar system [11] provides a framework for aggregating streaming context information in peer-to-peer pervasive computing networks, and [5] proposes a system of "sentient" objects that simplify the process of fusing and interpreting context information from multiple sources. Aggregation can itself be a means of improving efficiency, as seen in many sensor network applications where aggregating sensor readings can be used to reduce the overhead of sending redundant information to a single sink [14], [28], [30].

The sustained research interest over the past decade is a testament to the promise and power that context can wield when solving important mobile and pervasive computing application challenges. However, the field lacks a unified set of tools that excel in efficiently sensing, aggregating, and sharing context in a manner generalized for a wide range of real-world pervasive computing applications. Our group focuses on crafting context awareness tools that provide software engineers with the expressive and efficient building blocks they need to easily create applications that generate and consume context without requiring them to understand or worry about the underlying mechanisms.

The Time is Now. Advances in smartphone technology have elevated these platforms to a level that is sufficient for distributed and cooperative context-sharing applications. In fact a number of "distributed" context sensing mechanisms already exist commercially, for example in the guise of WiFi geo-location [2]. Many of these new applications fall under the guise of *participatory sensing*, which explicitly engages users in generating context information. However, despite the wide availability of peer-to-peer connections among these smart devices, if context is shared at all, it is only shared with centralized services.

It is our premise that unified and cooperative sharing of context will open up entirely new technologies like in-network caching of context information, cooperative mobile data offloading, intelligent pre-fetching using user mobility prediction, etc. Many of these ideas have been posited in ubiquitous and pervasive computing research, some as long as twenty years ago, but means to accomplish them within realistic overhead limitations remains a challenge. There are also a number of other challenges to wide-spread context sharing, privacy being the most often-cited. This could be partially solved by *user-centric context models* that leave the user in charge of what is shared and when [18], [38]. These approaches necessitate the design of intuitive user interfaces that expose appropriate settings without overwhelming users. Alternatively, in-network context aggregation can promote context privacy by blurring the data across many users before sharing it with the network.

III. Context Sensing and Sharing

The evolution and wide-spread integration of context awareness has been limited both by a lack of attention to the *efficiency* with which context can be acquired and by a lack of *expressiveness* in the resulting context information. Mobile and pervasive computing applications are heavily dependent on the availability and accuracy of contextual information. We have previously elicited *types* of context that could be collected in mobile pervasive computing environments and the types of context-based adaptation that could be accomplished given good contextual awareness [35]. Table I provides an overview of context types and their potential uses from our prior work; it is by no means exhaustive. Precious few of these metrics have good estimators or predictors available today, even though the sensors upon which these primitives must be

built are nearly ubiquitous. Part of the problem is battery life—accurate context awareness almost always comes at a cost of high battery consumption.

Achieving the goals of acquiring and adapting to the context types shown in Table I requires extending the reach of context beyond the simple egocentric views described in the previous section. In this section, we look at mechanisms to mitigate these concerns, focusing first on efficient ways to acquire context information that requires using more than just locally connected sensors. We then look at how it is possible to make *sharing* of context views possible, addressing both goals of efficiency and expressiveness. The former specifically because sharing of context information, especially across resource-constrained wireless networks, must be resource-conscious. We conclude with a discussion of how expressive and efficient sharing enables the aggregation of context from multiple entities to generate a broader view of context.

A. Passive Context Sensing: Efficient Context Acquisition

There are a number of ways to target efficient context sensing in resource constrained environments, however, there is always a trade-off between accurate context sensing and resource utilization. This is especially important in regards to network resources, which are expensive both in terms of battery usage required to send and receive wireless messages and the fact that communication relies on a limited commodity that many devices must share. Traditional mechanisms that extend context sensing beyond a device's own local sensors rely on *active* metrics, or metrics that generate additional network traffic in order to measure context (for example network latency) or at the very least exchange information such as location (for example to measure node mobility). However, much useful context can be measured through *passive* means through eavesdropping on existing network traffic. We have developed a framework, the Passive Sensing Suite, to gather passive context metrics; we also examined the correlation between the passively sensed context in the real world, and a ground truth estimation based on simulation [34]. We used our framework to examine several important context metrics including *network load*, *network density*, and *network dynamics*. The former two are self-explanatory; the latter attempts to capture the relative mobility between pairs of nodes. Although we found—unsurprisingly—that passive metrics are not as accurate as their actively sensed counterparts due in part on the reliance on existing network traffic, several metrics can be correlated (for example packet error rate and load) to increase the sensing accuracy. We found through experimentation that passively sensed metrics can be good estimators of their actively sensed counterparts.

B. Grapevine: Expressive and Efficient Context Sharing

In pervasive computing applications, the local context information on any given node can often prove useful to nearby nodes as well; however, such context is unavailable unless it is distributed. Indeed, in many pervasive computing applications, sending context information represents the vast majority of all communications. For example, an application that facilitates opportunistic activities, such as a serendipitous game of football among park visitors, is keenly interested in whether anyone nearby would also like to play. In such applications, actively sharing context information enables unique capabilities but must be extremely efficient to be feasible in any real system. We developed the Grapevine context dissemination framework to aide in the development of applications that actively consume context information from nearby nodes [16], generalizing the common aspects of sharing context so that developers can focus on the tasks that are unique to their application.

There are two key challenges in developing a framework with Grapevine's goals: the expressivity it offers developers in specifying the context information it shares, and the efficiency with which it shares it. To address the challenge of expressivity, we needed a means for application developers to identify individual context items that provide the maximum flexibility with a minimum of *a priori* requirements. To provide this we developed Grapevine so that context information is tracked using the familiar abstraction of a *mapping* from *keys* to *values*. A producer of context provides a unique identifier along with a value for that context item. Consumers can check the value for any given context item by querying Grapevine with the appropriate identifier. This high-level abstraction provides applications a large degree of freedom in choosing identifiers that can range from simple alphanumeric strings to more complex hierarchical/tuple-based identifiers. Furthermore, as long as the generation of the identifiers is consistent between nodes, they can be generated dynamically at runtime, avoiding the limitations of requiring static identifiers determined *a priori*.

For efficiency, we turn to probabilistic data structures. The most well known probabilistic data structure, the Bloom filter, allows for the membership of a set to be encoded with significant space savings over traditional set data structures [6]. The savings come with the tradeoff that false negatives will occur with a configurable probability (e.g., occasionally the filter will indicate that a member is in the set even though it was never added)[1]. However, false negatives are not possible (e.g., the filter will never indicate that an item is not in the set when it actually was added). Grapevine uses Bloomier filters, a variant with similar properties and construction that allows a value to be associated with each member of a set [10]. This capability mimics the mapping abstraction described above while providing a very space efficient representation.

Grapevine uses this expressive and efficient representation of context as the foundation of its context dissemination strategy. Each Grapevine-enabled node forms an individual context summary that contains the context information it is willing to share. These summaries also include a hop-count that specifies how many hops away it believes the context information might prove useful (thus limiting context dissemination to some set

[1]False positives can be mitigated by optimizing data structure properties, but a discussion of this is beyond the scope of this paper.

Type of Context	Examples	Usage
System Context	battery level, charging status, CPU load, free memory	selectively enable/disable a client's participation in mobile caching and ad-hoc content sharing
Network Context	network type, roaming status, calling status, WiFi state	can influence sharing patterns; enables content *prediction*, which is required for advanced delivery
Location Context	GPS location, speed, heading	can provide common mobility patterns for prediction; correlated with cache availability
Aggregate Context	common activities, social connections	servers can learn popular locations for caches and popular data to cache
Data Context	creation time, time-to-live, data size, priority labels	influence which data can be off-loaded depending on the network cache capabilities

TABLE I
POTENTIALLY USEFUL CONCRETE CONTEXT METRICS

of local neighbors). Nearby nodes periodically share their individual context summaries along with the summaries they have received from any other neighbors that have not yet traveled beyond the summary's internal hop count[2]. Sharing summaries in this manner gives each node access to the context information of nearby nodes.

This strategy has proven effective in systems where all the nodes are interested in the same set of context information; however when interest in context information differs between nodes, it quickly becomes inefficient because context information is being transmitted that no nearby node is interested in receiving. To improve efficiency in these situations, Grapevine provides an interest tracking capability that allows nodes to indicate the context information they wish to receive. Efficiently tracking this interest offers yet another challenge, leading to the development of a new probabilistic data structure variant, a spatiotemporal bloom filter (SpTBF, or "spitty bif") tailored for this purpose [17]. SpTBFs allow each node to keep track of the context information that neighbors recently indicated interest in receiving by storing a single local data structure that is attached to each Grapevine communication. This data structure maps the entire set of possible context identifiers to a smaller set of entries in an array of 2-tuples. False positives are again possible since multiple context items are mapped to each tuple, but the probability of these false positives can be tuned by adjusting parameters such as the array size. Each of these tuples tracks both a spatial component that decays as the interest-tracking SpTBF is passed around and a temporal component that decays as time elapses. To indicate interest in a context item, a node sets the tuple values for that context item to the amount of spatial and temporal permanence it desires for that interest—this controls persistence. Before sending context summaries, a node checks its local interest-SpTBF to see if anyone nearby is interested in the context information it has available and will only send that piece of context information if the tuple values for each context item are non-zero.

With these mechanisms for expressive and efficient context sharing in place, a natural next step is to allow individual context summaries to be aggregated into summaries that contain context information from multiple sources. This aggregation allows for a new notion of the shared context of groups and enables additional efficiency while simultaneously extending the reach of context information. Returning to our pick-up football game scenario we find that a list of individuals interested in playing can be aggregated into a single group summary. Similarly, the locations of many nodes can be aggregated into a bounding box that defines the region they occupy. By aggregating context, the applications no longer need to receive individual summaries from each participant and can instead receive a single aggregated summary containing the information they require. This concept of aggregation opens the doors to an exciting new area of research where the context of the group enables applications otherwise not possible. We continue to explore these group dynamics as an active and ongoing research topic.

[2]The hop count is decremented each time the summary is shared, and dissemination is squelched when the count reaches zero

IV. A STRAWMAN: THE CONTEXT AGENT FRAMEWORK

In this section, we bring together the conceptual perspectives from the previous section and motivate a lightweight and general purpose framework that supports context sensing, sharing, and adaptation. As a strawman for discussion, we present our *Context Agent Framework*, which was originally motivated by our prior work using context to inform routing decisions in opportunistic ad-hoc networks [33]. The context agent is a general systems framework for context aggregation and context-based adaptation; its scope is broad and embodies several concepts:

- the context types of interest cannot be entirely known *a priori*; any universal context solution must consider dynamic typing to remain flexible to new context types;
- broad categories of context exist (system, data, user, network, etc.), and they each have different aggregation strategies; a universal context framework must allow for all such strategies to coexist;
- adding new context types, collection, aggregation and sharing mechanisms, and implementing context-based adaptation should be straightforward and simple; and

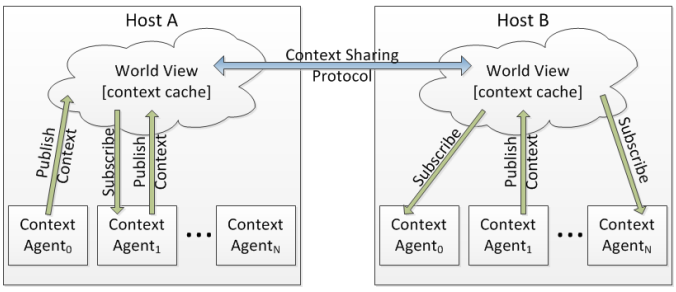

Fig. 1. Context Agent Framework Architecture

- a universal context framework should support multiple programming abstractions, allowing for ease of use, as well as context sharing across multiple nodes to allow for automatic context distribution.

These concepts motivate our *Context Agent Framework* (CAF), a modular and flexible framework for collecting, aggregating, sharing, and adapting to context. CAF embodies our new perspective on context, which combines both egocentric and shared context while supporting efficient and expressive context-awareness. CAF is a multi-threaded architecture comprising one or more *Context Agents* that either "produce" context updates (by sensing, aggregating, or both), or "consume" context to produce context-based system adaptations; a given context agent can be both a producer and a consumer of context. Fig. 1 shows a high-level view of the CAF architecture. The framework is designed to run as a privileged *user-space* process, and each element of the architecture is embodied by one or more threads within the framework. This was done to provide concurrency among multiple context agents collecting independent context samples, as well as to allow blocking within the implementation of any given agent. The following describes the elements of our framework.

Context Agents. The purpose of the Context Agent is two-fold: (1) gather context from the system, user, or network and (2) use context to adapt application behavior. The passive context sensing approaches described in the previous section can be incorporated into passive context sensing agents in the CAF architecture. In general, CAF is intended as a framework for quickly, and easily developing new context agents rather than an exhaustive set of all useful context agents. There are two basic types of agents *listeners*, and *gatherers*. As the name implies, listener agents implement a server process that waits for incoming connections from outside sources (e.g., sensors) to provide context. CAF supports any kind of blocking service implementation, although in our implementation we have limited ourselves to TCP sockets. Naturally, the listener must understand the exact format of the incoming context to be able to parse it, and this format must be agreed upon before CAF is instantiated. The gatherer implements a proactive agent that fetches context. The means by which a gatherer can gather context is limited only by the possibilities available to a privileged user-level process. For example, a gatherer context agent could read battery life by calling a battery monitoring daemon, by reading the appropriate node in the Linux `proc` filesystem directly, or even by opening a file that stores battery life samples taken in the past. The main difference between gatherers and listeners is that gatherers proactively fetch context, usually on a timer-based trigger. Both types of agents can post context to the CAF's *World View* (described in more detail below), where it will be stored and made available to any local agents that subscribe to the type of context.

The second purpose of Context Agents is to use context to adapt application behavior. Essentially this can be thought of as evaluating some *context-adaptation function* using the available context in the World View to generate some output—the output is then used to inform some system change to tweak the behavior. Once again, the is intended to be open-ended in regards to how exactly this is accomplished. Since each agent runs as a separate thread within CAF, context-based adaptations themselves can be anything a privileged, user-space process can accomplish—in our experiments, we have limited ourselves to adapting the parameters of routing protocols, but there is no limit to the possibilities. Similarly to context aggregation, context-based system adaptation can be accomplished by any combination of system calls, file or socket writes, inter-process communication, etc. The re-evaluation of the adaptation function can be triggered in two ways; either with a timer, or on a context sample update.

Context Formats and Types. CAF is purposefully open-ended in regards to the formatting of context samples. However, all context samples must conform to a loose standard based on the tuple concept [15]. Every sample must be associated with a type, which is itself just string representation. A context sample is formatted as: $\{type{:}timestamp{:}value(s)\}$. For example, location might be encoded as $\{\textbf{location} : timestamp : longitude : latitude\}$. There are two general classes of context available in CAF: local context and global context. Local context is relevant only to the local system and potentially to nearby neighbors. Examples might include a device's battery life, or a user's interest in a pick-up game of football. Global context is context that is intended to be shared across the entire network. In reality, global context can be shared only among nodes that meet, so there is no guarantee of coverage. Global context types are also generally tagged with the geographic location at which they were sampled. Examples might include geo-tagged node density estimates or geo-tagged network congestion estimates. The size of these geographical areas is user-configurable, and is discussed further on.

World View. The *World View* acts as the local context cache. As the CAF collects and processes context, it generates a synthesized view of the salient context types in the operating environment. This world view consists of (generally geographically tagged) global context samples stored in a dynamically generated "map" of the network. This map is split into user-configurable sized cells, and each cell contains a collection of context tuples (i.e., $\{type{:}timestamp{:}value(s)\}$) that correspond to that location. Context tuples that are not tagged with a location are stored in a single non-geographically-associated container. To accomplish context distribution, every node periodically shares its world view with its neighbors using

an efficient encoding such as the bloomier filter employed by Grapevine. When a node receives the world view of another node the two are merged according to a merge algorithm. In practice, this merge algorithm is generally simply replacement (the sample with the latest timestamp is the one that is kept) but could easily be a more sophisticated aggregation like those described above, including a rolling average, or some other similar operation. This sharing and merging of views allows CAF to provide applications an approximate view of the global context.

The World View supports two main operations, *publish*, and *subscribe*. Publish operations allow context agents to add new context samples to the World View (analogous to the **out()** operation of tuple spaces). Subscribe operations allow agents to register their context interests with the World View, and in doing so receive updates; this is analogous to reactive capability common to many tuple space implementations. In subscribing to context, an agent can pass in a list of types, even type wildcards, and that agent automatically receives updates to any sample, or any new samples, that conform to the type. Contrary to traditional publish/subscribe systems, our publish and subscribe primitives are only available locally—an agent can only subscribe to context updates generated in its own local World View. It cannot subscribe to updates from another node's World View. This limitation exists because of the unpredictable nature of wireless network links, which cannot be predicted or relied upon with any level certainty. However, since context samples are spread across the network by means of the World View sharing and merging capabilities, agents on one node *can* and do receive context samples generated on other nodes.

V. Adapting to Context

While uses of egocentric context have been explored in existing applications, the ability to efficiently and expressively sense aggregate forms of context and further share them with other nearby entities enables new uses of context. In this section, we explore a few of these concretely, showing how each new use is related to our new perspective on context.

A. Context-Based Adaptation of Network Coded Routing

In our prior work, we have used the CAF to adapt routing protocols for delay-tolerant networks (DTNs). Such networks are characterized by high mobility, poor connectivity, and long (minutes to hours, even days) delays in packet delivery. Routing is generally accomplished through probabilistic store-and-forward strategies, since end-to-end connections are rarely, if ever, available to a pair of communicating nodes. The general routing strategy for such networks involves creating multiple copies of data to improve the probability of eventual delivery. Network coding, one of the more recent and promising routing techniques to emerge for DTNs, has the potential to both reduce overhead and delivery latency through the mixing of packets from multiple sources [42]. The probabilistic mixing produces linearly independent combinations of packets; this mixing, among other things, improves the chances that any given communication will contribute "novel" information to the network [40].

We have adapted a network coded DTN routing protocol [36] using context to further optimize the bandwidth usage. In short, influencing routing decisions using context centered on *information diversity* has given us up to 300% improvements in delivery latency over non-context aware network coded implementations [33]. Our context, information diversity, measures the total information available in various geographically separate portions of a DTN. Mobile nodes take information diversity samples as they move though the network, and they share the samples through the *World View* element as described in Section IV. The routing protocol can then use this context, combined with information about the intended waypoints of mobile nodes to prioritize how packets are sent across the channel to maximize the increase in overall information diversity of the network.

B. Context-Based Adaptation of Mobility

Shared context information can also play an important role in adapting the navigation patterns of mobile platforms. In several recent experiments, we explored how the CAF-like capabilities of our Grapevine context dissemination framework simplified development of mobile applications. These experiments used the Pharos Testbed, a collection of general purpose x86-based computers with access to modular sensors (e.g., infrared proximity, GPS, etc.), a standard 802.11b network connection, and mounted atop a rugged mobility plane.

In one series of experiments, we developed software to patrol a perimeter consisting of fixed waypoints, deploying multiple coordination strategies to evaluate which was best at spacing the robots evenly along the perimeter. Two of these strategies are interesting for their use of context. In the first, we created a custom communication protocol that allowed a node to inform the node in front of it that it had arrived at a given waypoint. Robots would not proceed until they received a notification that the trailing robot had reached it's waypoint. We then implemented the same functionality using the Grapevine framework, which vastly simplified the required code. Instead of a program that required configuring sockets and handling a custom protocol, the Grapevine-based code simply included its current waypoint within its context summary and created a local world view that allowed it to monitor the summaries it received from other nearby nodes. Using the world view perspective, a robot could learn when another robot arrived at the trailing waypoint [16].

Building upon this, we are currently evaluating an application that eschews manually configured waypoints altogether. Instead, in these experiments we deploy multiple robots that each use their sensors to search for a specified target (e.g., a person wearing a bright pink shirt). This low-level sensor data is provided as context information to Grapevine, which allows the the group of patrolling robots to aggregate it into an accurate assessment of where the target is along with the number and location of patrolling robots. All of this context information is fused to dynamically determine a perimeter

around the target and inform each robot as to where it should go to best patrol that perimeter. Furthermore, since each node is aware of the locations of other all the other robots, mobility can be adjusted to ensure that collisions are avoided. Sharing and gathering the context information involves using extremely simple primitives, allowing for much easier development of such applications.

C. Context-Based Data Offloading for Cellular Networks

We have also used network, data, and location context to design an intelligent mobile data offloading web service architecture [35]. Intelligent data offloading moves data from the over-burdened cellular networks (usually 3G or 4G networks) onto to higher bandwidth but shorter range WiFi networks without degrading the user experience. Our design splits web content among multiple independent delivery vectors based on context. The key issues in designing our architecture, dubbed MADServer, were *(i)* what context to acquire and *(ii)* how to distribute and respond to the acquired context intelligently. Using concepts from the Context Agent Framework, we prototyped a web service architecture that relies on context to inform its data offloading decision, splitting responses to client web requests into two parts: $Response'$, to be delivered using the existing cellular wireless network, and $Response''$, to be delivered to a context cache location provided by the client. The dual delivery vectors allow us to provide uninterrupted connectivity to any given web service, even as the user moves between content caches and WiFi access areas promoting a frustration-free user experience, while at the same time saving precious cellular bandwidth by moving "heavy" content over the cheaper, higher bandwidth offloading vector. Our prototype showed promising results both in terms of cellular bandwidth savings and in content delivery speeds.

Similar architectures will be more widespread in the future as cellular bandwidth becomes an increasingly precious commodity. The growth in demand of cellular spectrum is far exceeding the increase in availability, and offloading is one of the more promising solutions. However, "smart" and efficient offloading relies not only on distributed content caches but also on efficient and expressive context awareness. The web services must know *where* to send the data, and *when*; in a world of mobile devices, this is not an easily answered question. Context aggregation and sharing architectures such as the Context Agent Framework are a key step in realizing the kind of context-awareness that intelligent mobile data offloading requires.

VI. Future Directions

In this paper, we have promoted a shifting perspective on context and context-awareness for mobile and pervasive computing. This shift has been motivated by both technological advances that make varying types of context information more prevalent and widely-available, and applications that demand shared views of group context situations. This shift in perspective opens many new avenues for intellectual pursuits.

One of the most obvious directions relates to privacy. A first observation comes from recent studies of aspects of information sharing in pervasive computing environments [22], [31]. Such studies have providing enlightening insight into people's willingness to share information; specifically, people may be willing to share with other co-located users (even unknown ones) what they would not share publicly (i.e., on the Internet). This motivates our perspective on *local* sharing, in which context information is not posted to a centralized source but instead is shared locally through a peer to peer network. In many instances, however, simply restricting sharing to locally connected peers will not be enough to maintain adequate privacy. Future research must study how entities can share information, even locally, without exposing sensitive information linked to users' identities. One promising direction is to explore the interplays of *trust*, *information aggregation*, and *privacy*.

Other privacy controls may rely on *user-centric* models that explicitly allow users to control what is shared when and with whom. Of course exposing this control to the user opens up obvious user interface concerns; it is not yet evident what the right abstractions are for communicating context privacy concerns to users, and developing these abstractions will be essential to mediating user-centric models.

As we sense context information from a wider variety of sensors and share that context information more broadly, questions naturally arise relating to the *quality* of context information. Furthermore, as we aggregate information with the goal of maintaining some degree of privacy, we may intentionally add *noise* to otherwise perfect data. These concerns motivate new metrics for context quality that are particularly sensitive to the joint notions of efficiency and expressiveness as well as to aggregation and sharing.

Finally, the dynamics of context are also related to these notions of quality. Given that we have aggregated and shared context views, how do we maintain them, updating them with new information as it becomes available and timing them out when they have effectively expired. This is especially challenging when considered jointly with privacy; if a context value has been aggregated or obfuscated to protect the entity that contributed the value, it is not clear how to handle an update to an already shared value. This is further complicated by the fact that the environment of sharing and aggregating is inherently distributed and subject to unpredictable network mobility and disconnection. Even finding the right values to update proves challenging.

We argue that a unified view of context—combining egocentric perspectives with distributed global perspectives—is the right direction and will be a key step in alleviating these kinds of problems. Our own recent efforts in adaptive, shared perspectives have yielded positive results. Next generation mobile and pervasive computing environments will depend on the kind of holistic view such an approach can provide.

REFERENCES

[1] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Cooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, 3:412–433, 1997.

[2] J. Angwin and J. Valentino-Devries. Apple, google collect user data. http://online.wsj.com/article/SB10001424052748703983704576277101723453610.html, 2011.

[3] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[4] P. Basu, N. Khan, and T. Little. A mobility based metric for clustering in mobile ad hoc networks. In *Proceedings of the 21st International Conference on Distributed Computing Systems Workshops*, pages 413–418, 2001.

[5] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications*, pages 361–365, March 2004.

[6] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Comm. of the ACM*, 13(7), 1970.

[7] P. Bolliger and B. Ostermaier. Koubachi: A mobile phone widget to enable affective communication with indoor plants. In *Proceedings of Mobile Interaction with the Real World*, pages 63–66, 2007.

[8] W. Brunette, R. Sodt, R. Chaudhri, M. Goel, M. Falcone, J. Van Orden, and G. Borriello. Open data kit sensors: A sensor integration framework for android at the application level. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pages 351–364, 2012.

[9] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. Srivastava. Participatory sensing. In *World Sensor Web Workshop*, pages 1–5, 2006.

[10] D. Charles and K. Chellapilla. Bloomier filters: A second look. In *ESA*, 2008.

[11] G. Chen, M. Li, and D. Kotz. Data-centric middleware for context-aware pervasive computing. *Pervasive and Mobile Computing*, 4(2):216–253, April 2008.

[12] A. Dey and G. Abowd. Cybreminder: A context-aware systems for supporting reminders. In *Proceedings of the 2nd International Conference on Handheld and Ubiquitous COmputing*, pages 172–186, 2000.

[13] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction*, 16(2–4):97–166, 2001.

[14] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wireless Communications*, 14(2):70–87, April 2007.

[15] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[16] E. Grim, C.-L. Fok, and C. Julien. Grapevine: Efficient situational awareness in pervasive computing environments. In *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications (Work in Progress)*, pages 475 –478, March 2012.

[17] E. Grim and C. Julien. Spitty bifs are spiffy bits: Interest-based context dissemination using spatiotemporal bloom filters. Technical report, The University of Texas at Austin, 2012.

[18] A. Gupta, M. Miettinen, and N. Asokan. Using context-profiling to aid access control decisions in mobile devices. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications (Demonstrations)*, pages 310–312, March 2011.

[19] G. Hackmann, C. Julien, J. Payton, and G.-C. Roman. Supporting generalized context interactions. In T. Gschwind and C. Mascolo, editors, *Software Engineering and Middleware: 4th International Workshop, Revised Selected Papers*, volume 3437 of *Lecture Notes in Computer Science*, pages 91–106, 2005.

[20] C. Hartung, A. Lerer, Y. Anokwa, C. Tseng, W. Brunette, and G. Borriello. Open data kit: Tools to build information services for developing regions. In *Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development*, 2010.

[21] J. Hong and J. Landay. An infrastructure approach to context-aware computing. *Human Computer Interaction*, 16(2–4), 2001.

[22] Q. Jones, S. Grandhi, S. Karam, S. Whittaker, C. Zhou, , and L. Terveen. Geographic place and community information preferences. *Computer Supported Cooperative Work*, 17(2–3):137–167, 2008.

[23] C. Julien. The context of coordinating groups in dynamic mobile environments. In *Proceedings of the 13th International Conference on Coordination Models and Languages (Coordination)*, pages 49–64, June 2011.

[24] T. Jun and C. Julien. Automated routing protocol selection in mobile ad hoc networks. In *Proceedings of the 25th ACM Symposium on Applied Computing*, pages 906–913, March 2007.

[25] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. SeeMon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, 2008.

[26] A. Karypidis and S. Lalis. Automated context aggregation and file annotation for PAN-based computing. *Personal and Ubiquitous Computing*, 11(1):33–44, December 2007.

[27] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

[28] L. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, pages 575–578, 2002.

[29] F. Lin, A. Rahmati, and L. Zhong. Dandelion: A framework for transparently programming phone-centered wireless body sensor applications for health. In *Proceedings of Wireless Health 2010*, pages 74–83, 2010.

[30] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *ACM SIGOPS Operating Systems Review*, volume 36, December 2002.

[31] D. Mountain and A. MacFarlane. Geographic information retrieval in a mobile environment: evaluating the needs of mobile individuals. *Journal of Information Science*, 33:515–530, 2007.

[32] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd International Symposium on Wearable Computers*, pages 92–99, 1998.

[33] A. Petz, A. Hennessy, B. Walker, C.-L. Fok, and C. Julien. An architecture for context-aware adaptation of routing in delay-tolerant networks,. In *Proceedings of the 4th Extreme Conference on Communication*, 2012.

[34] A. Petz, T. Jun, N. Roy, C.-L. Fok, and C. Julien. Passive network-awareness for dynamic resource-constrained networks. In *Proceedings of the 11th IFIP International Conference on Distributed Applications and Interoperable Systems*, 2011.

[35] A. Petz, A. Lindgren, P. Hui, and C. Julien. Madserver: An architecture for opportunistic mobile advanced delivery. In *Proceedings of the ACM MobiCom Workshop on Challenged Networks*, 2012.

[36] A. Petz, B. Walker, C.-L. Fok, C. Ardi, and C. Julien. Network coded routing in delay tolerant networks: An experience report. In *Proceedings of the 3rd Extreme Conference on Communication*, 2011.

[37] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 434–441, 1999.

[38] M. Tiwari, P. Mohan, A. Osheroff, H. Alkaff, E. Shi, E. Love, D. Song, and K. Asanovic. Context-centric security. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Security*, August 2012.

[39] N. Villar, J. Scott, and S. Hodges. Prototyping with Microsoft .NET Gadgeteer. In *Proceedings of the 5th International Conference on Tangible, Embedded, and Embodied Interaction*, pages 377–380, 2011.

[40] B. Walker, C. Ardi, A. Petz, J. Ryu, and C. Julien. Experiments on the spatial distribution of network code diversity in segmented dtns. In *Proceedings of the ACM MobiCom workshop on Challenged Networks*, 2011.

[41] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, page 179, 2009.

[42] J. Widmer and J.-Y. L. Boudec. Network coding for efficient communication in extreme networks. In *Proc. of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, 2005.