# Radiator - Context Propagation based on Delayed Aggregation

**Pedro Alves**
INESC-ID / Technical University of Lisbon / Opensoft
Rua Joshua Benoliel, 1, 4C, 1250 Lisboa
pedro.alves@opensoft.pt

**Paulo Ferreira**
INESC-ID / IST / Technical University of Lisbon
Rua Alves Redol, 9, 1000 Lisboa
paulo.ferreira@inesc-id.pt

## ABSTRACT

Context-aware systems take into account the user's current context (such as location, time and activity) to enrich the user interaction with the application. However, these systems may produce huge amounts of information that must be efficiently propagated to a group of people or even large communities while still protecting the privacy of the participants.

We argue that both scalability and privacy can be ensured by delaying context propagation until certain conditions are met and then aggregating such messages both at the syntactic and semantic level. Since such conditions vary from application to application, we propose Radiator, a systematic way to model the propagation characteristics of a distributed context-aware system.

Our qualitative evaluation shows that Radiator is generic enough to model the needs of different context propagation scenarios. To assess the impact of the model on the scalability of an application, we developed twiRadiator, an adaptation of Twitter to the Radiator model which, while preserving user expectations, reduces bandwidth consumption to approx. one third.

### Author Keywords

Context propagation; privacy; scalability

### ACM Classification Keywords

H.4.3 Information Systems Applications: Communications Applications

### General Terms

Algorithms; Human Factors; Measurement; Performance; Security

## INTRODUCTION

Context-aware systems take into account the user's current context (such as location, time and activity) to enrich the user interaction with the application [12, 34]. In the last decade, this topic has seen numerous developments that demonstrate its relevance and usefulness, a trend that was further accelerated with the recent widespread availability of powerful mobile devices (such as smartphones) that include a myriad of sensors which enable applications to capture the user environment for a large number of people. Recently, some commercial applications have started to make use of this contextual information to provide real-time traffic information (Waze[1]) or universities' hotspots (Bonfyre[2]), not to mention Facebook or Twitter whose personal short messages are complemented with identity, time and location.

These applications can operate on different scales depending on their purpose: personal, group or community [25]. Whereby personal context-aware applications are designed for a single individual (e.g., personal finance), group applications are designed to propagate context among a group of people who share a common goal or concern (e.g., avoid interruptions when calling friends [32]). When attaining such goals is only possible with a large number of people, applications start operating at a community scale (e.g., traffic monitoring [20] or noise map of a city [33]).

This distinction is important because, as we move from personal to group and then to community applications, we have to face increasing problems related to scalability and privacy. In fact, it is undoubtedly much easier to propagate context to a dozen of friends than to thousands of strangers (obviously, at the individual level this is not even a concern). For example, as of 2012, there are 175 million tweets (twitter messages) being sent per day and some of these messages are distributed to over 19 million users (the number of followers of Lady Gaga).[3] Foursquare[4] has millions of users sharing their location on a regular basis, with more than 600.000 updates per day.[5] Also, we are much more sensitive in revealing personal information to such large groups than we are to our close group of friends. For example, several people have been arrested or fired because of messages they posted on Facebook.[6]

Traditionally, application designers have decided to apply completely different mechanisms to the different scales. Firstly, community applications usually anonymize all information while group applications rely on trust between group members to prevent privacy breaches. Secondly,


*CSCW '13,* February 23–27, 2013, San Antonio, Texas, USA.


[1] http://www.waze.com
[2] http://www.bonfyreapp.com/
[3] See http://bit.ly/zOiX8k.
[4] http://www.foursquare.com
[5] http://blog.foursquare.com/2010/05/17/607883149/
[6] http://www.huffingtonpost.com/2011/08/30/arrested-over-facebook_n_942487.html

group applications propagate context in intact form while community applications aggregate context from multiple people into single statistical measures. Thirdly, context propagation is immediate on group applications while community applications usually imply some propagation lag to prevent scalability problems. Finally, community applications usually present the information in a condensed form to avoid overwhelming the user with the sheer amount of captured information, while this is not a problem on group applications. These differences are summarized in Table 1.

| Group | Community |
|---|---|
| Members trust each other | All data is anonymized |
| Context intact | Context aggregated |
| Immediate propagation | Delayed propagation |
| User sees all context info | User sees only aggregated info |

**Table 1. Differences between group and community applications**

However, we believe these boundaries will tend to blur in the future, as the distinction between personal, group and community goals also start to blur. As an example, consider the case of weight loss programs assisted by a context-aware application. These applications track user's weight on a daily basis as well as her diet, what she has eaten, how many times she went to the gym, etc. The main goal is clearly individual but it's a well known fact that a group of overweight people can achieve better results if they work together.[7] On top of that, there is statistical relevant data from large communities that can help in the weight losing process, such as knowing that consuming a certain ingredient is usually related to weight losing from a statistical point of view. So, weight losing programs could benefit from the three scales but, since the mechanisms mentioned in the previous paragraph are usually applied to only one scale, application programmers usually end up choosing only one scale instead of embracing a *multi-scale* approach.

There are several other types of applications suffering from this problem. Typically, "friends location" applications are designed at group scale but if they introduce features such as "popular places" they start operating at community scale. These applications are constantly propagating context between groups of friends (i.e., their location, current activity, etc.) to improve coordination between them, promote serendipitous encounters, etc. However, since they are already receiving this contextual information, they might as well discover the most popular spots (e.g., pubs, restaurants) based on the number of people located there (regardless of being friends). In Table 2 we present more examples of this type of applications.

Even when developing applications for just one scale, application programmers spend a significant amount of time designing and programming the context propagation layer, a relatively generic component which could be reused on multiple applications. Some frameworks have been developed to provide a common layer to context-aware applications, but they are usually focused on abstracting the sensor layer (e.g., Context Toolkit [34]) or rely on a publish/subscribe model of propagation (e.g., PACE [21]) which lacks two important features for community-level applications (see Table 1): delayed propagation and anonymization.

Our goal is threefold: to propose a model that (1) is generic enough to express the context propagation and privacy needs of distributed context-aware applications[8] that are able to produce context information (according to Definition 1 outlined in the Design Section); (2) improves scalability by substantially reducing the amount of transmitted information while still fulfilling user expectations; (3) provides a privacy mechanism out-of-the-box without imposing any effort from neither the application programmer nor the user.

We address these challenges by proposing Radiator, a context propagation model that operates equally well on the different scales of context-aware systems and effectively addresses scalability and privacy. This model relies on four main principles:

- Context messages have different urgency levels for different recipients. Some recipients will tolerate (and even appreciate) some lag on the propagation of certain context messages that they do not consider very important. Therefore, some messages can be retained before being propagated.
- Messages that are being retained can be aggregated prior to their propagation. This aggregation can be a simple compression or more sophisticated semantic aggregations. In any case, the final aggregated message is normally much smaller than the sum of the original individual messages. This brings two advantages: network bandwidth reduction (therefore, improved scalability) and less information overload for the end user.
- Anonymization can be achieved through proper aggregation: by aggregating sensible data in such a way that it is not possible to match that data with the corresponding individuals while still retaining its statistical usefulness.
- Application programmers should be able to define their context propagation specific needs with a simple yet expressive model that maps seamlessly to an easily pluggable middleware component.

The remainder of this paper is organized as follows. The next section surveys relevant related work. Then we present the Radiator model, a very simple yet expressive model to define different context propagation schemes. The next section describes a middleware that implements the Radiator model and that can be easily plugged by programmers into their context-aware applications. Then, we evaluate our three goals: generality, scalability and privacy. Finally, we draw some conclusions.

## RELATED WORK

In this section, we review the literature for the usage of aggregation, privacy protection and context propagation techniques on context-aware systems.

[7] Take for example the TOPS (Take Off Pounds Sensibly) organization which is based on weekly support meetings between its (more than 200.000) members.

[8] Applications where context is captured in one place and consumed in another place, therefore requiring context propagation across some network.

### Aggregation

In distributed context-aware applications, aggregation may be performed at two levels: the inference/reasoning level and the distribution level.

Aggregation at the inference level provides application developers with data on a higher level of abstraction [4] and is usually materialized on some kind of semantic aggregation. For example the Activity widget provided by the Context Toolkit [34] transforms raw audio data from a microphone into "Activity Level" values (*none*, *some* or *a lot*). Many examples like this exist in the literature (e.g., [29, 38, 35]) but they usually perform the aggregation next to the sensor creating a strong coupling that prevents reuse. For example, inferring a street name from a pair of geo-coordinates is so common that it makes sense to move that logic into the middleware layer. The Solar system [8] presents a middleware where aggregation can be composed of multiple operations parametrized by each application. However, it is not able to retain messages during a certain period of time, which would increase the efficiency of the aggregation.

Aggregation at the distribution stage occurs when context was already captured and processed and is now in the process of being sent to the client application. This aggregation consists of gathering into a single large message the content of multiple smaller messages before propagating it. Even though the vast majority of the context-aware frameworks propagate context in single disaggregated messages [1], recent experiments with aggregation have achieved good results [3, 15]. In both cases, we're talking about syntactic aggregation: blindly concatenating single packets into a larger one and possibly compressing it. Although this yields good compression levels for unstructured text-based messages, we believe a higher-level semantic aggregation would be much more effective with other types of information (e.g., geo-localized messages or sensor readings). Ideally, the developer should be able to provide an aggregation function suited to his application needs that would be executed by the middleware.

### Privacy

Distributed context aware applications may expose personal information such as current location or availability to a large group of people, raising privacy concerns on end-users. The increasing pervasiveness of devices that are able to communicate context information without explicit user intervention is only aggravating this problem.

In general, privacy mechanisms can be grouped into four categories [2]:

- **Privacy policies** - Applications using this technique allow the user (the context discloser) to provide rules (privacy policies) that define to whom and to what extent his context information is revealed to others. This is the most common technique on both academic and industry social applications (e.g. Facebook, Twitter) and it is tied to *control* and *accountability*, two important mechanisms for privacy protection. However, this approach has revealed multiple problems: it is cumbersome for users to specify fine-grained policies and users are not particularly good at it [11]. For example, Gross [19] found that only 1.2% of Facebook users changed the default privacy settings for profile search-ability.
- **Data perturbation** - This type of technique consists of transforming or partially omitting information before being delivered to the context consumer, in such a way that it is impossible to reconstruct the original message while still keeping (some of) its usefulness [17]. Usually, this is implemented through encryption [31], noise addition [17], blurring [24] or chunk replacement [30]. These techniques are most effective for community-scoped applications whose primary purpose is gathering statistical information.
- **Anonymization** - Using this type of technique, the information is delivered intact to context consumers, except for its author, which is removed or replaced, preventing an attacker from inferring the real author [7, 14]. The removal process must affect both directly identifiable attributes and indirectly identifiable attributes, also known as quasi-identifiers [18]. Some examples of anonymization techniques are mix routing [10], temporal-spatial cloaking [20] and hitchhiking [37]. Unlike the other types of technique, anonymization can usually be parametrized and measured using well-known metrics such as *k-Anonymity* [36], *l-Diversity* [28] and *t-Closeness* [26]. The *k-Anonymity* metric is based on the idea of generalizing a data record until it is indistinguishable from the records of at least $k$ - 1 other individuals. For example, this idea can be applied to location information [20] by computing geographical areas large enough to contain $k$ locations (and during a certain period of time), and propagate that area instead of the individual locations.
- **Lookup notification** - This technique consists on providing the user with information of who has consumed his context information and when [9]. This can occur in real-time (the user is alerted that someone is consuming his context information) or *a posteriori*, by keeping a log of who has seen which information. Unlike the previous three techniques, which are applied before data is delivered, this technique is applied after data delivery (that is, after a potential privacy breach) so it is usually combined with other techniques.

Despite extensive research on this subject, the community is still far from reaching a complete solution that covers the privacy needs of context-aware applications. Two major concerns are the rigid nature of most approaches and the inability to cover *multi-scale* applications (i.e., applications that work both at group scale and community scale). Consolvo [9] has shown that privacy settings can't be rigid - they must be situation-dependent. However, *privacy policies* and *data perturbation* techniques are usually applied regardless of the situation. On the other hand, *data perturbation* techniques are tied to community-scale applications (given their statistical bias) while *lookup notification* techniques are usually applied to group-scale applications (they don't scale well - the user would be overwhelmed with thousands of notifications on community-scale applications).

Anonymization techniques offer a promising solution to those concerns because they usually depend on a parameter that

can be configured for each case (therefore being situation-dependent) such as, for example, the $k$ or the $l$ in the k-Anonymity/l-Diversity privacy metrics. Furthermore, they are easy to implement, don't depend on the user's ability to define privacy rules and actually reflect real-world social behavior (e.g., people are more comfortable with video-vigilance on crowded places than on deserted places).

Most of the context-aware frameworks don't offer any support for privacy. Some exceptions such as Context Fabric [23] and PACE [21] allow applications to manage a repository of privacy policies whose rules are enforced upon access to information. However, as already mentioned, this approach relies on users specifying their rules, which has been proven not to be very effective. Also, to the best of our knowledge, none of the public context-aware frameworks support parametrized anonymity mechanisms that can be measured using *k-Anonymity* or *l-Diversity* metrics.

### Context Propagation

Several context-aware frameworks rely on traditional publish-subscribe approaches to selectively propagate context messages (e.g., PACE [21], CFN [8] and AWARE [5]). On publish-subscribe systems, users subscribe to topics (subject-based systems) or predicates (content-based systems) [16]. Then, users feed content into the system (publish) and the system distributes events matching subscribers interest with publisher content. Even though the decoupled nature of this approach makes it attractive for systems that need to address mobility and heterogeneity requirements, it assumes a relatively fixed set of matching rules. Systems where context attributes change very frequently (e.g., location, speed) require large volumes of subscribe/unsubscribe messages, leading to wasted resources and poor scalability.

## DESIGN

This section describes Radiator, a generic context propagation model based on the concept of aggregation. Radiator assumes a distributed system where a group of personal sensors (sensors that capture the situation of a person) want to propagate context messages to others, regardless of their number (i.e., it works for both group-level and community-level propagation). In most cases, each person is both a producer and a consumer of context messages, but the underlying model is general enough to work on systems where producers are different than consumers. Additionally, this model assumes that context messages will always be propagated albeit some of them can be delayed and/or transformed.

Note that it is up to the application programmer to setup a model that suits the needs of her application. Even though the programmer may take into account the user's input, it is not the user's responsibility to setup Radiator.

First, we need to define context as a triplet relating people, time and the attributes that characterize the environment or situation of those people during a given time span.

**Definition 1 (Context)**

*Let an attribute $A_i$ to be a tuple $(N, V)$, where N is a name representing the attribute (e.g., speed) and V is the value of that attribute (e.g., 100); $P$ to be a finite set of people $\{P_1, P_2, .., P_n\}$; $t$ to be a time range between two timestamps $t_i..t_j$; $A$ to be a finite set of attributes $\{A_1, A_2, .., A_n\}$.*

*Context $C$ is a triplet ($P$, $t$, $A$) that represents the attributes that characterize the situation of a group of people $P$ during the time span $t$.*

For example, suppose Alice is in New York between Jul 1st and Jul 3rd. We can define her context $C$ as:

$C = ((`Alice'), `01/07'..`03/07', ((`location', `NewYork'))$ .

The Radiator model also introduces the notion of context aggregation to represent a set of contexts $C$ that share a certain attribute $A$, producing an aggregated context $C_{Aggr}$.

**Definition 2 (Aggregation)**

*Let $\{(P_1, t_1, A_1), .., (P_n, t_n, A_n)\}$ be a set of contexts and $A_c$ a common attribute such that $\forall A_i \in A_1...A_n$ : $A_c \in A_i$.*

$Aggr(A_c, \{(P_1, t_1, A_1), .., (P_n, t_n, A_n)\}) = (P, t, A) \Rightarrow$
$\forall P_i \in P_1..P_n : P_i \in P \wedge$
$\forall t_i \in t_1...t_n : t_i \subseteq t \wedge$
$\forall A_i \in A_1...A_n : A_i \in A$

For example, it is possible to aggregate:

$C_{Alice} = ((`Alice'), `01/07'..`03/07', ((`location', `NewYork'))$
$C_{Bob} = ((`Bob'), `02/07'..`04/07', ((`location', `NewYork'))$

into an aggregated context:

$C_{Aggr} = ((`Alice', `Bob'), `01/07', `04/07'], ((`location', `NewYork'))$

or (if anonymity is required):

$C_{Aggr} = ((`A', `B'), `01/07', `04/07'], ((`location', `NewYork'))$

Any function capable of aggregating contexts according to definition 2 is noted as $f_{Aggr}$. Aggregation can be done at the raw data level (e.g. compression) or at the semantic level, taking into account the specific properties of certain attributes. For example, if the attribute is a geographical location, a common $f_{Aggr}$ is the minimum bounding box - the smallest rectangle where a set of coordinates are included. Other common $f_{Aggr}$ calculates the mean for continuous attributes or the median for categorical attributes [13].

Finally, we introduce the notion of aggregability.

**Definition 3 (Aggregability)**

*Let $C_P$ be the current context of a given person $P$ and $C_x$ the context of someone else that the system wants to propagate to $P$. The aggregability function $G(C_P, C_x)$ represents how much aggregated $C_x$ must be before being transmitted to $P$, taking into consideration his current context $C_P$.*

*Therefore, a set of contexts $C_1$ to $C_n$ is only propagated to $P$ when $\forall i \in 1..n, G(C_P, C_i) = n$, $n$ being an integer.*

Informally, the aggregability represents the number of context messages that must be retained before propagation. If we define an aggregability function ($G$) that always returns 4, the system will always aggregate four context messages before propagating them. In case we want immediate propagation, we can define $G$ as a function that always returns 1. Although returning a simple number has the advantage of keeping the model very simple, we propose that $G$ returns a tuple in the following format:

$G(C_P, C_i) \rightarrow \{type : value\}$

$type :: [volume|time|people]$

Along with the before mentioned $value$, $G$ must tell if that value represents a quantity ($volume$), a time range ($time$) or the number of different people contained in the aggregation ($people$). If the type is $time$, context messages will be aggregated until the number of seconds between the oldest and newest retained message is equal or greater than $value$. The types $volume$ and $people$ are similar in the fact that they represent the maximum number of retained/aggregated messages, although $volume$ is the number of different messages while $people$ represents the number of different people involved on those messages. For example, if $G$ returns $\{people : 4\}$, the system will aggregate messages until there are four different people involved, before propagating them. This distinction is important to implement privacy management mechanisms as we will describe shortly. Note that we chose these 3 metrics as the minimal set to fulfill the context propagation needs of most distributed context-aware applications (as shown in the Evaluation Section); other metrics can easily be included without changing Radiator's algorithm.

Given these definitions, we can now describe how the Radiator model works. A set of people ($P_1..P_n$) wants to propagate their current context ($C_{P_1}..C_{P_n}$) to everyone, in an efficient and privacy-manageable way. Every context $C$ is evaluated against a set of aggregability functions $G$ that will tell how many other "similar" $C_n$ must be aggregated with $C$. If that aggregation is not possible (i.e., there are not enough aggregatable pending $C_n$), than $C$ is appended to a *pending* queue. On the other hand, if there are conditions to aggregate, we can apply an aggregation function $f_{Aggr}$ to those contexts that produces a single aggregated context $C_{Aggr}$. Afterwards, $C_{Aggr}$ is propagated and the corresponding $C_n$ removed from the pending queue.

This model is much more efficient than traditional immediate broadcast approaches because it postpones propagation until a certain level of aggregability is met. Since the aggregability function reduces a set of context messages into a single one, it effectively reduces the number of bytes transmitted over the wire. This is particularly significant when the message is propagated to a large population, as the message size reduction is multiplied by the number of recipients. Moreover, it provides a simple yet powerful mechanism for privacy management based on the concept of k-anonymity [36]. The k-anonymity principle tells that we can achieve anonymity by aggregating at least *k* records with a common attribute, making them indistinguishable from each other. Implementing k-anonymity using this model is straightforward - the developer only has to define an aggregability function $G \rightarrow (\{people : k\})$. Note that the model allows other anonymity metrics such as l-diversity [28] (e.g., using an aggregability function such as $G \rightarrow (\{diverse\_salary : l\})$) or t-closeness [26]. We only implemented support for k-anonymity because of its simplicity and generalized adoption.

As an example, let's say we want to propagate only context information related to at least two different people. Therefore, we define an aggregability function
$G \rightarrow \{people : 2\}$.

A typical message flow would be:

t:0 Alice wants to propagate $C_{Alice_1}$. Since there are not enough messages to aggregate with, it's appended to the *pending* queue.

t:1 Alice wants to propagate $C_{Alice_2}$. There is already one message to aggregate with, but it's from the same person, so it's also appended to the *pending* queue.

t:3 Bob wants to propagate $C_{Bob_1}$. We can aggregate this message with the other pending messages and get an aggregated context of at least two people. So we apply $f_{Aggr}(C_{Alice_1}, C_{Alice_2}, C_{Bob_1})$ to get $C_{Aggr}$. $C_{Aggr}$ is propagated to everyone.

Until now, we have considered only constant aggregability functions, that return the same value independently of the actors involved. However, consider the case where we want to have different propagation rules depending on whether people are friends or strangers. For example, a certain application may want to immediately propagate a person's context to her friends but aggregate messages up to 40 seconds when they are being propagated to strangers. We could define such function as follows:

$G(C_P, C_{P_i}) \rightarrow \{volume : 1\} \iff is\_friend(P_i, P)$
$G(C_P, C_{P_i}) \rightarrow \{time : 40\} \iff is\_stranger(P_i, P)$

We present other examples of aggregability functions as well as an evaluation of their generality in the Evaluation Section.

## IMPLEMENTATION

We developed a middleware component that implements the Radiator model described in the Design Section, in order to (1) provide a reference implementation for developers who wish to use this model on their applications and (2) conduct several experiments to evaluate the model regarding ease of use and performance/scalability. The results of the latter are presented in the Evaluation Section.

The middleware component is written in Python and it can be downloaded from `https://bitbucket.org/palves/radiator`. From a client perspective (a developer who wishes to integrate her context-aware application with the Radiator middleware), there are only two classes to know about: `Context` and `RadiatorMiddleware`. The class `Context` contains all the information pertaining to a certain context message according to Definition 1: people, time and a list of attributes. The `RadiatorMiddleware` class is the primary interface from which to send and receive context messages (represented as Context objects). The constructor receives an aggregability function that complies with

| # | Scenario | Description | Aggregability function |
|---|---|---|---|
| 1 | Traffic monitoring | Aggregate speedometer and GPS data within 300 seconds periods | $\{time : 300\}$ |
| 2 | Road hazards detection | Aggregate vertical accelerometer and GPS data until 5 hazards detected | $\{volume : 5\}$ |
| 3 | Popular spots Friends location | Aggregate location until 10 different people in the same spot but for friends send immediately (non-aggregated) | $\{people : 10\}$ if stranger $\{volume : 1\}$ if friend |
| 4 | Facebook likes | Aggregate likes from strangers within 300 seconds periods, likes from friends of friends until there are 5 and likes from direct friends with a maximum delay of 30 seconds | $\{time : 300\}$ if stranger $\{volume : 5\}$ if friend_of_friend $\{time : 30\}$ if friend |
| 5 | Friends location in crowded spaces (concerts, street markets) | Aggregate location based on how far you are from the recipient (further away implies more aggregation) | $\{volume : \texttt{distance}\}$ |
| 6 | Stock market alerts | Aggregate stock market information during a period of time proportional to the number of shares owned by the recipient (higher number implies less aggregation) | $\{time : 1000/1 + \texttt{number\_of\_shares}\}$ |
| 7 | Weight loss program | Anonymize weight information based on the recipient's weight (the closer the weight of recipient is from the sender the less anonymous it is) | $\{people : \texttt{weight\_difference}\}$ |
| 8 | Anonymize twitter | Anonymize tweets with Egyptian revolution related hashtags | $\{people : 50\}$ if `#sidibouzid` or `#Jan25` |

**Table 2. Different context propagation scenarios and the corresponding aggregability functions**

```
1  def my_aggregability(recipient, context):
2      # aggregates until 4 hazards detected or
3      # 5 minutes since the last propagation
4
5      if context.attributes['hazard_detected']:
6          return {'volume': 4}
7      else:
8          return {'time': 300}  # 300 seconds
9
10 def mean(attribute_name, attributes_list):
11     # returns the mean value of attributes named
12     # 'attribute_name' in 'attributes_list'
13     ...
14
15 # start the middleware engine
16 engine = Middleware(aggregability=my_aggregability,
17                     aggregation_functions={'hazard_detected':mean,
18                                            'speed':mean })
19
20 # register some participants
21 engine.register('p1')
22 engine.register('p2')
23
24 # propagate a context message
25 engine.propagate(Context('p2', now, { 'speed' : 100 }))
26
27 def receive(context):
28     # called whenever a context is received.
29     # the context may be aggregated.
30
31 # register the callback for receiving messages
32 engine.register_receiver(receive)
```

**Listing 1. Code example of Radiator middleware usage**

Definition 3 and an optional list of aggregation functions (Definition 2), as show on lines 16-18 in Listing 1. Each participant then registers itself (lines 21-22) after which it can start propagating messages as well as receiving messages propagated by others.

The rest of the process is also exemplified in Listing 1. After the `RadiatorMiddleware` initialization, the application can propagate context (line 25) or can receive context that was propagated by others (lines 27-29). Since the reception of context is asynchronous, the developer has to register a callback to do so (line 32).

Note that the actual network topology of the system is completely transparent to the developer. Even though right now this is implemented following a centralized model, it can easily accommodate other models such as the partitioned or the decentralized topologies without breaking the client application.

## EVALUATION

The Radiator model is simple enough to be easily understood by application developers yet sufficiently expressive to define a wide range of needs usually found on distributed context-aware applications. In this section, we first measure its expressiveness by applying the Radiator model to eight scenarios where context propagation across the network is crucial to accomplish the application's goals. Note that, although evaluating the usability of this model (for developers) was not the goal of this paper, our experiments have given good indications on this subject.

Then, we present *twiRadiator*, a Twitter application built over the Radiator model to evaluate the impact of different propagation rules on (1) the system scalability and (2) user expectations. With over 100M users, Twitter is arguably one of the largest context-propagation applications and has faced multiple scalability problems along the time[9].

Finally, we evaluated how the Radiator model can protect the privacy of Twitter participants in sensitive situations such as the Egyptian revolution.

### Is the model generic enough?

Despite the simplicity of the Radiator model, it is still powerful enough to describe a large variety of context propagation scenarios. Consider the scenarios and corresponding aggregability functions on Table 2. For example, traffic monitoring applications [20, 22] (first row of Table 2) continuously capture the speed and position of a large group of vehicles to identify places suffering from traffic congestion. This information is then broadcast to all vehicles possibly alerting

[9]See http://tcrn.ch/wi7rJ3

| # | Scenario | Aggregation function |
|---|---|---|
| 1 | Traffic monitoring | `avg`(speed) by location |
| 2 | Road hazards detection | `sum`(hazards) by location |
| 3 | Popular spots | `count`(people) by location |
| 4 | Facebook likes | `sum`(likes) by object |
| 5 | Friends location in crowded spaces | `list`(people) by distance |
| 6 | Stock market alerts | `newest`(values) by share |
| 7 | Weight loss program | `list`(people) by weight |
| 8 | Anonymize twitter | `list`(people) by tweet |

**Table 3. Different context propagation scenarios and the corresponding aggregation functions**

the drivers to take alternative routes. Since traffic congestion occurs during a relatively long period of time (at least, dozens of minutes), it can be aggregated within 5 minutes (300 seconds) periods without losing its usefulness and relevance. Using the radiator model, implementing this propagation scheme is as simple as defining an aggregability function that always returns {`time:300`}.

The aggregation itself is performed by the *aggregation function* (see Definition 2 in the Design Section). If the developer does not provide any aggregation function, Radiator applies a simple concatenation between the messages to aggregate. To achieve higher compression levels (and therefore reduce network bandwidth), the developer should provide an aggregation function that takes into account the specific needs of its application. Table 3 shows some examples of such functions. In the traffic monitoring case (first row in Table 3), we are concerned about the average speed within a geographical region. For example, if the average speed is near zero, it is reasonable to assume that there is traffic congestion within that particular region. The pseudo-code of such function is shown in Listing 2.

```
def average_speed_by_location(pending_msgs):
  return [people_within_location,
          time_range_between_oldest_and_newest,
          { 'speed': avg(speed), 'location': location}
          for location in group_by(pending_msgs, location)]
```

**Listing 2. Pseudo-code of an aggregation function for traffic monitoring applications**

Since the propagation scheme is continuously calculated by the aggregability function, is is possible to define more complex scenarios such as *Facebook likes*[10] (the fourth row of Table 2). Depending on the privacy settings, when a user likes a link or a photo on facebook, this action can show up on the news feed of all his friends, friends of friends and even strangers. This is important to measure the general popularity of an item when compared to the popularity within the much narrower circle of friends. Again, in the radiator model this is very simple to define. In this case, the aggregability function returns three different settings depending on whether the recipient is a friend, a friend of a friend or a stranger. In the latter, it can aggregate *likes* within 300 seconds because we only want a coarse-grained statistical value ({`time:300`}). On the other hand, we want to see *likes* from our friends with some urgency (we may want to interact with them based on their *like*), so the aggregation is much narrower (only 30 seconds - {`time:30`}). For friend of friends, we may want to set the maximum number of retained/aggregated *likes* to only five ({`volume:5`}). Here, the aggregation function can just be the sum of likes grouped by liked object (see fourth row of Table 3), so that users know the current "popularity" of a given object (e.g., 5 likes for photo X, 23 likes for article Y).

Even though the previous case is more flexible, we are still limited to a discrete set of options (friend, friend of friend, stranger). However, the radiator model allows completely continuous propagation schemes, giving rise to potentially very creative models of propagation. Rows 4, 5 and 6 of Table 2 exemplify such creative uses. For example, typical weight loss programs where users keep track of their diet on a daily basis, as well as their current weight, are very sensitive to privacy concerns [17]. On the other hand, it is a well known fact that engaging with other users on similar conditions can have a positive effect on the program, blending the user's personal goal with the community goals (e.g., Fitbit[11]). Therefore, we suggest an application that aggregates weight information proportionally to the weight difference between the sender and the recipient. Suppose Alice weights 95 Kg. If there is another user elsewhere with the exact same weight, the application shows this information to Alice unaggregated (and vice-versa). But if Bob and Carol weight 94 Kg, their weight will appear aggregated to Alice (Alice will receive $('2people', 'Today', (\{`weight' : `94'\}))$. The aggregation (and consequentially the anonymity level) increases as the weight becomes further from 95 Kg. Using the radiator model, this scenario can be defined by an aggregation function that returns a result that is itself dependent on a variable (in this case, the weight) - $\{people :$ `weight_difference`$\}$.

With respect to privacy, we can turn to scenarios like the 2011 Tunisian and Egyptian revolutions that were greatly propelled by twitter, through the use of hashtags such as `#sidibouzed` (home city of Mohamed Bouazizi, following his self-immolation) or `#Jan25` (after a political demonstration in Cairo on the January 25th) [27]. The big challenge here lies in preventing the ones who start the tweets (typically, local activists) from being identified by the government in order to censor and control the dissemination of critical rebel information. Even though twitter allows the use of pseudonyms, given enough historical information one could still identify the author. Using Radiator's *k-Anonymity* style mechanism, it is possible to preserve the privacy of the actors involved by aggregating batches of messages until there are at least $k$ people involved. This rule could be applied only to sensitive hashtags, as in the example shown on the last row of Table 2.

#### Delayed aggregation impact on scalability

In the Radiator model, some messages can be aggregated before being propagated. Since the compression of an aggregated message is usually much more effective than the compression of its individual parts, we evaluated this effect

[10] See http://on.fb.me/wrw9bA for information on the like feature

[11] http://www.fitbit.com

| Scenario | Rel. Consumed Bandwidth | Max delay (secs) |
|---|---|---|
| $\{volume : 1\}$ | 100% | 0 |
| $\{volume : 5\}$ | 51% | 49 |
| $\{volume : 10\}$ | 43% | 62 |
| $\{volume : 20\}$ | 39% | 66 |
| $\{volume : 50\}$ | 35% | 144 |
| $\{volume : 100\}$ | 32% | 250 |
| $\{volume : 200\}$ | 29% | 471 |

**Table 4. Consumed bandwidth under different scenarios**

on the consumed network bandwidth of an application built using the Radiator middleware described in the Implementation Section.

We developed twiRadiator, an adaptation of the well-known Twitter application to the Radiator model. Twitter[12] is a very popular application with over 100M users, that allows anyone to post short messages with less than 140 characters (called "tweets") with diverse content, ranging from personal thoughts to photo and link sharing. Even though Twitter may not be considered a classic context-aware application, tweets are supposed to be answers for the question *"What's happening?"*, so we are effectively talking about a huge context propagation system and therefore an excellent candidate to test the impact of the Radiator model.

```
# start the middleware engine
engine = Middleware(['p1'], # 1 participant
            aggregability=[lambda p, c: {'volume': 10}],
            aggregation_functions=
            {'tweet': aggregate_equal_tweets,
             'location': aggregate_if_all_the_same})

# iterate all the tweets in the sample
for tweet in tweet_sample:
        engine.propagate(Context(tweet))
```

**Listing 3. Sample of twiRadiator showing how easy it is to setup a testing scenario with the Radiator middleware**

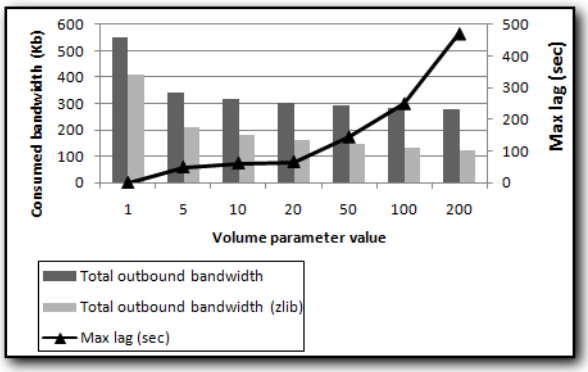


**Figure 1. Bandwidth consumption under different "volume" values and its effect on the message propagation lag**

We collected a sample of received tweets from a popular user (Scobleizer[13]) using Twitter's API. Scobleizer follows more than 32.000 users (i.e., receives tweets from more than 32.000 users). The sample contains all the received tweets by Scobleizer during 1 hour (on Jan 15th 2012), resulting in aprox. 2000 tweets (at least 1 tweet received every two seconds). We then fed this sample into twiRadiator, which transformed each tweet in a $context$ object and used the Radiator middleware to propagate those messages. Then, we measured the total consumed bandwidth for each user, under different $volume$ values. Listing 3 shows an example for these scenarios: a middleware engine is initialized with the $volume$ set to 10, meaning that the engine will postpone propagation until it can aggregate 10 messages.

We have also setup two **aggreggation functions**: one to aggregate tweets with the same content into a single one (basically a count of retweets) and another to aggregate locations if they were all the same (i.e., if all the tweets contained in the aggregation share the same location, that location is only included once). Here's a simple example of how 3 different messages with the same location become aggregated.

$([`p1'], (`04:00:00', `04:00:00'),$
$(`tweet' : [`strange'], `loc' : [`NewYork']))$

$([`p2'], (`04:00:02', `04:00:02'),$
$(`tweet' : [`This is great'], `loc' : [`NewYork']))$

$([`p3'], (`04:00:05', `04:00:05'),$
$(`tweet' : [`OMG'], `loc' : [`NewYork']))$

are aggregated into:

$([`p1', `p2', `p3'], (`04:00:00', `04:00:05'),$
$(`tweet' : [`strange', `This is great', `OMG'], `loc' : [`NewYork']))$

We experimented with the following $volume$ values: 1 (immediate propagation, equivalent to what Twitter does), 5, 10, 20, 50, 100 and 200 with zlib compression turned off and on. Note that, in this test, we are only propagating messages to one participant. This means that the **server**'s outbound bandwidth consumption (the focus of our experiment) equals the (single) **client**'s inbound bandwidth consumption.

From the results, shown in Figure 1, we can see that the consumed bandwidth decreases as the volume increases, even when compression is turned off. The reason is that in the aggregated messages we are not repeating the headers and other structural information that ends being repeated in every message. When we turn on the compression, the effect on the consumed bandwidth is even more clear, becoming more efficient as the number of aggregated messages increase. Comparing the compressed bandwidth used in the direct propagation scenario ($\{volume : 1\}$) with the most aggregated scenario ($\{volume : 200\}$), we can observe that the latter consumes only 29% of the former. However, we can also observe the impact on the maximum propagation delay as the volume increases. Since we are allowing the middleware to aggregate up to 200 messages before propagating, we can expect an increased delay. In the most aggregated scenario some messages would arrive with aprox. 8 minutes of delay, but if we look at the $\{volume : 50\}$ scenario, there seems to be a good tradeoff between the 2.5 minutes delay and the 35% bandwidth reduction (see Table 4). To better understand what would be a good tradeoff

[12] http://www.twitter.com
[13] http://www.twitter.com/scobleizer

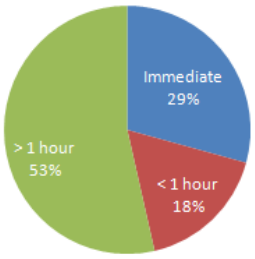


**Figure 2. Answers to the question "On Twitter, what is your maximum tolerable delay for receiving a tweet?"**

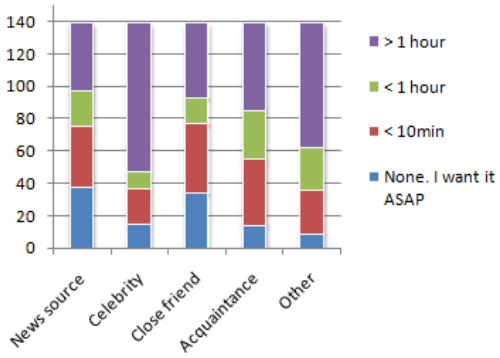


**Figure 3. Tolerable delay by type of source information**

from a user's perspective, we conducted a survey which we describe in the next section.

### Delayed propagation impact on user expectations

To better understand the user expectations of this type of applications we designed a questionnaire that was answered by 139 (non-paid) participants that were randomly recruited through twitter. All the subjects were active twitter users and 69% of them followed more than 100 users. We asked the following question:

*"On Twitter, what is your maximum tolerable delay for receiving a tweet from each type of source? That is, instead of receiving the tweet immediately, the tweet would arrive after a certain amount of time. How much time would you tolerate before becoming annoyed about it?".*

Respondents had to answer this question for each of the following five types of source information: News source (e.g., CNN); Celebrity (e.g., Lady Gaga); Close friend; Acquaintance; Other. The possible answers ranged from "None. I want it as soon as possible" to "I don't care, as long as I receive it." with some time ranges in between.

Our goal was to understand (1) if users tolerate receiving tweets with a delay and (2) if that delay depends on the type of source.

The general results without taking into account the type of source, depicted in Figure 2, show that only 29% of the respondents require immediate propagation and that the majority doesn't mind receiving tweets with over an hour delay. From that, we can conclude that twitter could use a delayed propagation approach while still fulfilling the needs of most of its users.

Next, we analyze if and how user expectations depend on the type of source. The results in Figure 3 show that the source of information is indeed an important factor on the maximum tolerable delay. Tweets from close friends and media sources require the most immediacy while tweets from celebrities require no immediacy at all. Given that the tweets from some celebrities are propagated to millions of users, the bandwidth reduction of delayed propagation that we observed on the quantitative evaluation could have a tremendous impact while still fulfilling user expectations. We can also observe that the tolerance to delayed messages is much greater for acquaintances and strangers than close friends. This seems to indicate that the distance on the social network graph is correlated with the delivery urgency of the corresponding messages. We believe this correlation is a common characteristic of distributed context-aware applications and fits perfectly into the Radiator model.

Given these results, the following aggregability function would match scalability needs with user expectations, assuming a function $social\_distance$ returning the number of hops in the social network between 2 people:

$\{time : 0\}$ `if media`
$\{time : 10000\}$ `if celebrity` # a few hours
$\{time : social\_distance(p1, p2) * 3600\}$ `if !(media or celebrity)`

The specific implementation of $social\_distance$ is outside the scope of this paper but we present some hints on possible solutions. The most obvious solution, albeit cumbersome, would be to ask the user directly about how strong is his relation with each person he follows. Another solution would be to tap into other social networks (besides Twitter) that support different levels of connections such as Facebook's friends and friends of friends and Linkedin's 1st, 2nd and 3rd level connections. Finally, even on Twitter itself we can distinguish between symmetrical relationships (A follows B and B follows A) and asymmetrical relationships (A follows B but B doesn't follow A) to infer the strength of the connection.

### Privacy

The Radiator model proposes a privacy mechanism based on the *k-Anonymity* principle: generalizing a data record until it is indistinguishable from the records of at least $k$ - 1 other individuals. To evaluate how this principle applies to real-world data, we collected a sample of tweets with the hashtag `#Jan25` pertaining the Egyptian revolution (after a political demonstration in Cairo on the January 25th). Egypt and several other countries have been reported to track down activists on social networks such as twitter and facebook[14], therefore privacy is a major concern on these scenarios.

In Figure 4 we can see a sample of tweets in its original form (using the Twitter web client). The tweets are shown in a way that it is clear who is their author, making it impossible to protect his privacy. To keep the figure simple, we only show the original tweets, not the retweets from 7 other people.

Next, we setup the Radiator model with the following configuration:

[14]http://rww.to/ifg3i0


**Figure 4. Sample of tweets with the hashtag #Jan25, in its original form**


**Figure 5. The same sample of tweets in aggregated form.**

$\{people : 12\}$ if `#Jan25`

This configuration tells the Radiator to aggregate messages with the hashtag `#Jan25` until at least 12 different people are included in the aggregation. We chose the number 12 to keep the example simple: a greater number would increase privacy even more. As shown in Figure 5, applying this configuration to the same sample of tweets, we get a result that protects the privacy of the participants without losing critical information. Now, the same messages appear next to each other, with all the authors in the end, so it is no longer possible to relate tweets with their authors. Even though some local activists may be included in the authors, we are no longer sure about the specific tweet they are responsible for. They may even claim that they were just retweeting a tweet from a foreign journalist, for example (which is probably enough to get by the government control). In practice, during the peak of revolution, it would be reasonable to use much higher levels of *k-Anonymity*, with $k$ easily going up to the thousands. Note that, in this example, we used an aggregation function that preserved the identity of the authors (although breaking the link between author and message). We could just as well used a function that replaced the identity of the authors with a count (number of authors), achieving greater privacy at the expense of some utility. The point here is that the Radiator model can be used to implement multiple levels of privacy with little effort by carefully selecting adequate aggregability and aggregation functions. For example, as suggested by Campan [6], we could take into account the social network of each participant to drive the anonymization level by using an aggregation function such as $\{people : num\_friends()\}$, where num_friends() is the number of friends of the author.

## CONCLUSIONS

In this paper, we propose Radiator, a simple and generic model for propagation characteristics of distributed context-aware systems. The model revolves around the concept of delayed aggregation: it allows application developers to define which messages should be propagated immediately and which should be retained and aggregated before being propagated. This definition is formally based on the concept of $Aggregability$, representing the conditions under which messages must be retained before propagation. These conditions may depend on *volume*, *time* or *people* and can be highly dynamic as the $Aggregability$ function may return itself a function.

As our experimental results have shown, it is possible to decrease network bandwidth consumption to less than 1/3 using the delayed aggregation technique (due to higher compression efficiency) while still fulfilling user needs. Furthermore, by carefully aggregating messages coming from different people in a way that makes it impossible to distinguish individual information (borrowed from the concept of *k-Anonymity*), it is also possible to protect the privacy of the participants, a common and important issue on the design of context-aware applications.

Thus, this model is able to improve both scalability and privacy with very little effort from the developer for a wide variety of applications, from traffic monitoring and location-based services to health monitoring and social network applications.

In the future, we plan to experiment with a decentralized Radiator middleware to improve the total consumed network bandwidth as well as implementing other applications on top of this model.

## ACKNOWLEDGMENTS

This work was partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/113993/2009 and PEst-OE/EEI/LA0021/2011.

## REFERENCES

1. P. Alves and P. Ferreira. Distributed Context-Aware Systems. Technical Report RT/22/2011, INESC-ID, 2011.
2. P. Alves and P. Ferreira. Privacy in Distributed Context-Aware Systems. Technical Report RT/23/2011, INESC-ID, 2011.
3. P. Alves and P. Ferreira. ReConMUC - Adaptable Consistency Requirements for Efficient Large-scale Multi-user Chat. In *Proceedings of the 2011 ACM conference on Computer supported cooperative work*, 2011.
4. M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263, 2007.
5. J. E. Bardram and T. R. Hansen. The AWARE Architecture: Supporting Context-Mediated Social Awareness in Mobile Cooperation. In *Proc. Conf. on Computer-Supported Collaborative Work (CSCW)*, pages 192–201, Chicago, 2004.

6. A. Campan and T. M. Truta. A Clustering Approach for Data and Structural Anonymity in Social Networks. *Privacy, Security, and Trust in KDD*, pages 0–9, 2008.
7. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
8. G. Chen, M. Li, and D. Kotz. Design and implementation of a large-scale context fusion network. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 246–255. IEEE, 2004.
9. S. Consolvo, I. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge. Location disclosure to social relations: why, when, & what people want to share. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 81–90. ACM, 2005.
10. C. Cornelius, A. Kapadia, and D. Kotz. Anonysense: privacy-aware people-centric sensing. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 211–224. ACM New York, NY, USA, 2008.
11. J. Cornwell, I. Fette, G. Hsieh, M. Prabaker, J. Rao, K. Tang, K. Vaniea, L. Bauer, L. Cranor, J. Hong, B. McLaren, M. Reiter, and N. Sadeh. User-Controllable Security and Privacy for Pervasive Computing. *Eighth IEEE Workshop on Mobile Computing Systems and Applications*, pages 14–19, Mar. 2007.
12. A. Dey and G. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 workshop on the what, who, where, when, and how of context-awareness*, volume 4, pages 1–6, 2000.
13. J. Domingo-Ferrer and V. Torra. Ordinal, Continuous and Heterogeneous k-Anonymity Through Microaggregation. *Data Mining and Knowledge Discovery*, 11(2):195–212, Aug. 2005.
14. M. Duckham and L. Kulik. Location privacy and location-aware computing. *Dynamic & mobile GIS: Investigating change in space and time*, pages 34–51, 2006.
15. J. Dyck, C. Gutwin, T. Graham, and D. Pinelle. Beyond the LAN: Techniques from network games for improving groupware performance. In *Proceedings of the 2007 international ACM conference on Supporting group work*, pages 291–300. ACM, 2007.
16. P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.
17. R. Ganti, N. Pham, Y. Tsai, and T. Abdelzaher. PoolView: stream privacy for grassroots participatory sensing. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 281–294. ACM, 2008.
18. P. Golle and K. Partridge. On the anonymity of home/work location pairs. *Pervasive Computing*, pages 2–9, 2009.
19. R. Gross and A. Acquisti. Information Revelation and Privacy in Online Social Networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80. ACM, 2005.
20. M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. *Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03*, pages 31–42, 2003.
21. K. Henricksen, J. Indulska, and T. McFadden. Middleware for distributed context-aware systems. *Internet Systems 2005*, pages 846–863, 2005.
22. B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson. Virtual trip lines for distributed privacy-preserving traffic monitoring. *Proceeding of the 6th international conference on Mobile systems, applications, and services - MobiSys '08*, page 15, 2008.
23. J. Hong and J. Landay. An architecture for privacy-sensitive ubiquitous computing. In *2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2004.
24. G. Iachello, I. Smith, S. Consolvo, M. Chen, and G. Abowd. Developing privacy guidelines for social location disclosure applications and services. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 65–76. ACM, 2005.
25. N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A Survey of Mobile Phone Sensing. *IEEE Communications Magazine*, (September):140–150, 2010.
26. N. Li, T. Li, and V. Suresh. t-closeness: Privacy beyond k-anonymity and l-diversity. *IEEE 23rd International Conference on Data Engineering*, pages 106–115, 2007.
27. G. Lotan, E. Graeff, M. Ananny, D. Gaffney, I. Pearce, and D. Boyd. The Revolutions Were Tweeted: Information Flows During the 2011 Tunisian and Egyptian Revolutions. *International Journal of Communication*, 5:1375–1406, 2011.
28. A. Machanavajjhala and D. Kifer. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), Mar. 2007.
29. E. Miluzzo, N. Lane, K. Fodor, R. Peterson, and H. Lu. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *6th ACM conference on Embedded network sensor systems*, page 337, New York, New York, USA, 2008. ACM Press.
30. M. Mun, P. Boda, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, and R. West. PEIR, the personal environmental impact report, as a platform for participatory sensing systems research. *Proceedings of the 7th international conference on Mobile systems, applications, and services - Mobisys '09*, page 55, 2009.
31. K. P. N. Puttaswamy and B. Y. Zhao. Preserving privacy in location-based mobile social applications. *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications - HotMobile '10*, page 1, 2010.
32. M. Raento, a. Oulasvirta, R. Petit, and H. Toivonen. ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. *IEEE Pervasive Computing*, 4(2):51–59, Apr. 2005.
33. R. Rana, C. Chou, S. Kanhere, N. Bulusu, and W. Hu. Ear-phone: an end-to-end participatory urban noise mapping system. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 105–116. ACM, 2010.
34. D. Salber, A. Dey, and G. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, page 441, New York, New York, USA, 1999. ACM.
35. T. Stiefmeier, C. Lombriser, D. Roggen, H. Junker, G. Ogris, and G. Tröster. Event-based activity tracking in work environments. In *Proceedings of the 3rd International Forum on Applied Wearable Computing (IFAWC)*. Citeseer, 2006.
36. L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.
37. K. Tang, P. Keyani, J. Fogarty, and J. Hong. Putting people in their place: An anonymous and privacy-sensitive approach to collecting sensed data in location-based applications. *Proceedings of the SIGCHI*, 2006.
38. E. Welbourne, J. Lester, A. Lamarca, and G. Borriello. Mobile Context Inference Using Low-Cost Sensors. In *Location and Context-Awareness*, pages 254–263. 2005.