REMINISCE: Transparent and Contextually-Relevant Retrospection

Sangsu Lee, Tomasz Kalbarczyk, and Christine Julien

The University of Texas at Austin
{sethlee, tkalbar, c.julien}@utexas.edu

***Abstract*—Many envisioned applications of pervasive computing, from smart personal assistants to interaction in the IoT, assume the ability to assess the context and provide data and services relevant to that context. As a demonstration of needed practical support for seamless context-based interaction, we present REMINISCE, an application that allows users to look back on their memories by prioritizing photographs that were taken in contexts that are similar to the current context. While we focus on the use of context to draw up photographic history, the vision behind REMINISCE is more general. We describe how REMINISCE is built on a contextual database middleware that provides context storage and context-sensitive queries that determine the similarity between pairs of contexts. REMINISCE uses the contextual database to store the contexts of a user's photos, which include location, time, neighboring devices, and weather conditions. Later, the app identifies the photos whose stored contexts are most similar to the user's current context by querying the contextual database; this query uses a context similarity function to compare entries in the database to the user's current context. During the demonstration, visitors will be able to interact with the app on the spot with auto-generated contexts and an on-device context simulation.**

I. INTRODUCTION

The proliferation of sensor-rich mobile devices and the advent of the Internet of Things entail applications that demand expressive context-awareness [9]. A context-aware device has the potential to support features such as envisioned in anticipatory computing [8]. In fact, it is increasingly common for mobile devices to provide information and services context-dependently like *Android Smart Lock*[1] and *Google Now*[2]. The REMINISCE app showcased in this paper will demonstrate yet another use case for this context, in particular, how context can be used to transparently recall information that is relevant in the user's current context. The primary objective of the REMINISCE app is to recommend and display the photos that are the most relevant to the user's current context as a way to encourage the user to look back on past events. The photo could be taken in the same place, at a certain time of the year, or with the same group of friends. In addition, REMINISCE may recall a photo taken in a same weather, as weather conditions are known to trigger memories [11]. While the REMINISCE app is playful, the more generic goal is not: the app's use of context is a stepping stone on the road to *in situ* context-based authorization, a holy grail of seamless interaction in the emerging Internet of Things [4], [3], [12]. Context-based authorization is a subset of access control that is specific to the mobile IoT setting, where attributes are permutations of context information. This work remains theoretical and without concrete mechanisms for deriving attributes from real-world context, a gap we seek to begin to fill with the implementation of REMINISCE. Penumbra [5] associates tags with user data to derive attributes and guide access control policies, but tags must be manually generated. Others have investigated mining attributes from existing access control logs [6], [14], but these rely on the existence of more primitive access control mechanisms (e.g., role-based access control). Other approaches seamlessly generate access control policies for limited use cases [2], [7] without explicitly deriving attributes. Without a mechanism for validating context, these approaches are susceptible to attackers spoofing contextual data to circumvent access protocols. REMINISCE serves as a prototype to demonstrate the first step in our approach to context-based authorization. It shows how we can utilize attributes that are derived from contextual data to help users transparently recall memories; in the future, these same attributes could be used to seamlessly generate fine-grained access control policies. We next overview the REMINISCE app, then describe the technical details of its implementation and close with a description of the intended demonstration.

[1]https://support.google.com/accounts/answer/6160273?hl=en
[2]https://en.wikipedia.org/wiki/Google_Now

II. OVERVIEW OF REMINISCE

To track the photos' contexts provide context-based recommendations, REMINISCE relies on a contextual database. This database, CONTEXTDB, maintains a mapping of user-generated content (i.e., photos) to context data; the latter is stored using a vector space model. The types of context the middleware supports are called *context attributes* [9]. When a new content item is created, the database gathers the current context information to associate with the new entry. In other words, every entry has a *context stamp* mapped to a content item in storage. Each *context stamp* can include multiple context attributes, all of which together capture the state of the world at the time the content item was created.

Our current version of CONTEXTDB stores the exact *context stamp* for each data item. In the future, we will explore lossy versions in which a given *context stamp* may be considered redundant if it is sufficiently similar to one already stored [13]. Instead, a new content item can be associated with

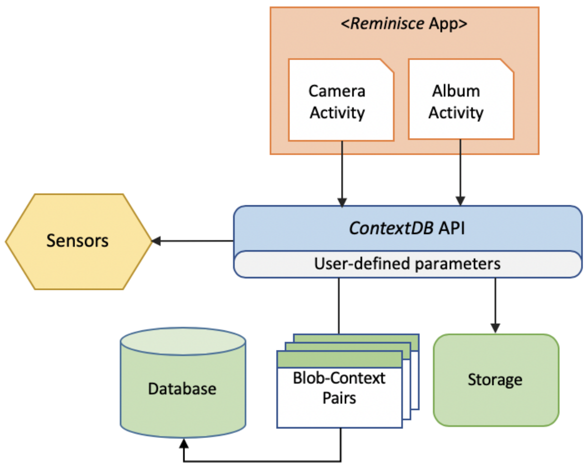


Fig. 1: Architectural Overview of REMINISCE

the previously captured *context stamp*. This will both reduce the storage overhead for CONTEXTDB and the query overhead for retrieving context matches from CONTEXTDB.

During a context query, an application specifies a set of context attributes to consider (i.e., a subset of those available). A query outputs pieces of content (i.e., photos) that were created in a context that is similar to the current one. Internally, REMINISCE relies on a measure of contextual similarity. A simple use case of this approach is to categorize and manage files based on the places in which the user accesses them (e.g., office and home); by incorporating more measures of the environment, one can envision implementing expressive context-based access control [1], [10].

III. IMPLEMENTATION

The implementation of REMINISCE is split into two pieces. The first piece, transparent to the user, is our CONTEXTDB, which is built specifically to maintain *context stamps* to support seamless queries from context-based applications. The second piece, visible to the user, is the REMINISCE app itself, which exemplifies an application able to leverage CONTEXTDB to seamlessly (with respect to user interaction) perform operations that rely on contextually-tagged data.

A. The CONTEXTDB Middleware

CONTEXTDB retrieves and processes raw contextual data and associates the context with a *data blob* such as a photo. Each context attribute is represented by a normalized context vector, $\hat{\mathbf{p}}_j = (x_1, x_2, \ldots, x_l)$, where l is a dimension of context attribute j and $x_k \in \mathbb{R}$. Context vectors are normalized so that the influence of each attribute can be treated equally when REMINISCE aggregates across attributes. Generating context vectors that are inter-operable despite different semantic meaning requires pre-processing and normalization steps that depend on characteristics of the raw attributes. For example, the context vector for a GPS location is derived by converting raw latitude and longitude into the Cartesian coordinate system such that x, y, and z are all normalized. Contexts like neighboring devices and weather can instead be denoted as bit vectors. For instance, $\mathbf{p}_{neighbors} = (1, 0, 1)$ indicates that of the three known potential neighbors, two are currently present. Multiple context vectors are joined to form a single *context stamp* that represents the device context at a fixed point in time. The *context stamp* for a given *data blob* is an aggregation of the context vectors for all m context types: $\boldsymbol{P} = \begin{bmatrix}\hat{\mathbf{p}}_1 & \hat{\mathbf{p}}_2 & \cdots & \hat{\mathbf{p}}_m\end{bmatrix}^T$. An instance of CONTEXTDB is then composed of blob-context pairs and can be denoted as:

$$\mathcal{D} = \{\langle d^{(1)}, \boldsymbol{P}^{(1)}\rangle, \langle d^{(2)}, \boldsymbol{P}^{(2)}\rangle, \ldots, \}$$

where the superscript is simply an index into the database.

A key requirement of REMINISCE is to determine how similar two *context stamps* are. The similarity of a pair of context vectors is defined by a function

$$s(\hat{\mathbf{p}}_j^{(i)}, \hat{\mathbf{p}}_j^{(c)}) = \hat{\mathbf{p}}_j^{(i)} \cdot \hat{\mathbf{p}}_j^{(c)},$$

where $\hat{\mathbf{p}}_j^{(c)}$ is a context vector in the current *context stamp*. The function satisfies the condition $-1 \leq s(\hat{\mathbf{p}}_j^{(i)}, \hat{\mathbf{p}}_j^{(c)}) \leq 1$. The output 1 indicates completely identical context vectors, while -1 denotes the lowest possible similarity.

Given a *context stamp* $\boldsymbol{P}^{(c)}$ capturing the current contextual state, the similarity to an entry $\boldsymbol{P}^{(i)} \in \mathcal{D}$ is:

$$S_c(\boldsymbol{P}^{(i)}, \boldsymbol{P}^{(c)}) = \frac{1}{m}\sum_{j=1}^{m} \omega_j s(\hat{\mathbf{p}}_j^{(i)}, \hat{\mathbf{p}}_j^{(c)}),$$

where m is the total number of attributes. Contextual similarity for individual attributes is weighted using a constant ω_j, where $\sum_{j=1}^{m} \omega_j = 1$. When a CONTEXTDB query uses only a subset of attributes of size m_s, to determine similarity, weights are normalized so that they satisfy $\sum_{j=1}^{m_s} \omega_j = 1$. In practice, ω_j reflects the relevance of attribute j.

Queries to CONTEXTDB consist of two inputs: a threshold for desired context similarity, τ, and a *context stamp* representing the current set of observable context attributes, $\boldsymbol{P}^{(c)}$. Based on these inputs, the query result, $\mathcal{Q}$, is defined as

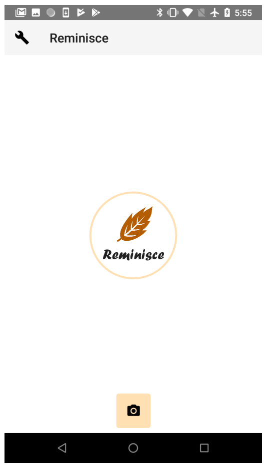
$$\mathcal{Q} = \{\langle d^{(i)}, \boldsymbol{P}^{(i)}\rangle \in \mathcal{D} \,|\, S_c(\boldsymbol{P}^{(i)}, \boldsymbol{P}^{(c)}) > \tau\}$$

The output consists of a set of *data blobs*, $d^{(i)}$, whose associated *context stamps* satisfy the similarity threshold.

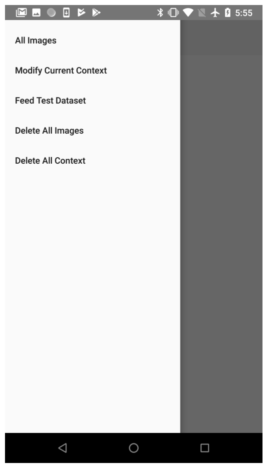
B. The REMINISCE App

To demonstrate the function of CONTEXTDB, we built the REMINISCE photo recall app[3]. For testing and demonstration purposes, REMINISCE includes the ability for the user to visualize all of their *data blobs* (i.e., photos) and their associated *context stamps*. While CONTEXTDB can handle generic context data, the REMINISCE app currently utilizes four context attributes: location, time, nearby devices, and weather. When the user takes a photo with the REMINISCE app or queries REMINISCE for photos similar to the current context, CONTEXTDB fetches contexts from on-device sensors. In particular, CONTEXTDB uses GPS and Android's location

[3]The REMINISCE app and the CONTEXTDB are available publicly: https://github.com/UT-MPC/Reminisce.

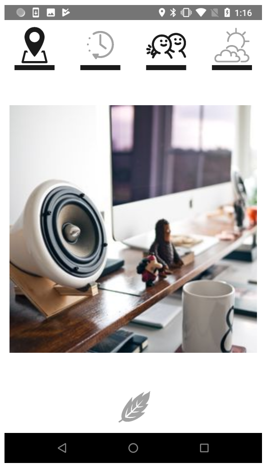


(a) A user can open album or camera from the main view

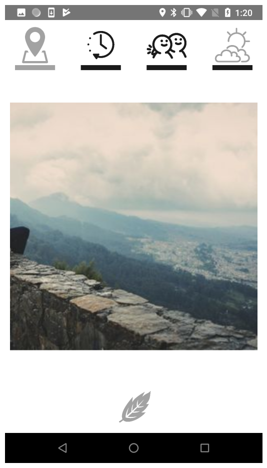


(b) REMINISCE provides a variety of testing capabilities

Fig. 2: Main window of REMINISCE



(a) Icons show which context matches the current context



(b) A context attribute can be enabled/disabled

Fig. 3: The album view of REMINISCE

services API to acquire the exact location. Weather context is obtained from an online service based on the location. Finally, CONTEXTDB uses Bluetooth Low Energy (BLE) beacons from nearby devices to derive the neighbor context.

In the main window of REMINISCE, a user may choose to take a photo or see an album (Fig. 2(a)). The icon on the upper left corner opens the menu for testing and demonstration (Fig. 2(b)). The button in the center of the main window leads to the album view, which is shown in Fig. 3.

REMINISCE's album view consists of icons that represent the four context attributes it supports, shown across the top. REMINISCE shows the stored photo that has the context most similar to the current context. The user can swipe left to see the photo with the next most similar context. As the user swipes through the stored photos, the icons associated with context attributes of the photo that have a similarity that satisfies $s(\hat{\mathbf{p}}_j^{(i)}, \hat{\mathbf{p}}_j^{(c)}) > \tau_j$ are highlighted. In Fig. 3(a), the icon depicting location is highlighted, indicating that the photo was taken at a location close to the user's current location.

The four icons are also buttons the user can toggle. When they are disabled, REMINISCE does not consider the corresponding context attribute. For instance, when the location context icon (the first icon from the left) is disabled (as shown in Fig. 3(b)), only the three remaining contexts are taken into account. Black or grey bars under the icons indicate whether the context attribute is enabled.

IV. DEMONSTRATION AND TECHNICAL REQUIREMENTS

Visitors will interact directly with the app. Visitors may experiment the app themselves with extra functionalities of REMINISCE by opening a navigation menu. The menu has a list of actions for testing the app. Using the testing menu, users can see a list of all photos, which also displays the context data for each photo when it is selected. Users can also modify the current context of the device, which will allow the user to simulate varying contexts without the user having to physically move. Visitors can also download test photos to seed the demonstration. The test photos will be stored in CONTEXTDB with random contexts relevant to the context of the conference. Lastly, contexts or photos can be deleted.

We will bring our own Android devices for the demonstration. However, any Android device with BLE support can run the app. A user has to allow permissions for location, camera, accessing data and phone state, and Internet.

REFERENCES

[1] M. Conti, B. Crispo, E. Fernandes, and Y. Zhauniarovich. Crêpe: A system for enforcing fine-grained context-related policies on android. *IEEE Trans. on Info. Forensics and Security*, 7(5):1426–1438, 2012.

[2] A. Gupta, M. Miettinen, N. Asokan, and M. Nagy. Intuitive security policy configuration in mobile devices using context profiling. In *Proc. of SocialCom*, pages 471–480. IEEE, 2012.

[3] V. C. Hu et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.

[4] J. Hur and D. K. Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Trans. on Parallel and Dist. Sys.*, 22(7):1214–1221, July 2011.

[5] M. L. Mazurek, Y. Liang, W. Melicher, M. Sleeper, L. Bauer, G. R. Ganger, N. Gupta, and M. K. Reiter. Toward strong, usable access control for shared distributed data. In *Proc. of FAST*, 2014.

[6] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari. Evolutionary inference of attribute-based access control policies. In *Proc. of EMO*, pages 351–365, 2015.

[7] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, and N. Asokan. Conxsense: Automated context classification for context-aware access control. In *Proc. of ASIACCS*, pages 293–304, 2014.

[8] V. Pejovic and M. Musolesi. Anticipatory mobile computing: A survey of the state of the art and research challenges. *ACM Comput. Surv.*, 47(3):47:1–47:29, Apr. 2015.

[9] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the Internet of Things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2014.

[10] G. Russello, M. Conti, B. Crispo, and E. Fernandes. Moses: supporting operation modes on smartphones. In *Proc. of SACMAT*, 2012.

[11] W. van Tilburg, C. Sedikides, and T. Wildschut. Adverse weather evokes nostalgia. *Personality and Social Psychology Bulletin*, 2018.

[12] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proc. of FMSE*, pages 45–55, 2004.

[13] N. Wendt and C. Julien. Paco: A system-level abstraction for on-loading contextual data to mobile devices. *IEEE Trans. on Mobile Comp.*, 2018.

[14] Z. Xu and S. D. Stoller. Mining attribute-based access control policies. *IEEE Trans. on Dependable and Secure Comp.*, 12(5):533–545, Sept 2015.