Pixelsior: Photo Management as a Platform Service for Mobile Apps

Kyungho Jeon, Sharath Chandrashekhara, Karthik Dantu, Steven Y. Ko
University at Buffalo, The State University of New York

Abstract

Photo management has become a sizable fraction of our computer interaction. Due to economic incentives, every software company wants to restrict users to using their software for photo management and use. Unfortunately, this results in duplication of images, repeated image manipulation operations, and an overall uneven and siloed user experience. In this paper, we motivate the need for a dedicated platform service for photo management which can not only manage the photos on one device, but also transparently manage content adaptation, image manipulation and propagation of the manipulation to all the applications on a device, and all devices using the service. Pixelsior presents our study of the requirements of such a system as well as a preliminary design motivated by requirements of consistency and efficiency.

1 Introduction

Spurred by the widespread use of camera-equipped devices, the number of photos taken everyday has been rapidly increasing over the last decade. While it is estimated that about 80 billion photos were taken in 2000, in 2014, 550 billion photos were shared on Facebook, WhatsApp and Snapchat alone [5]. With the advent of newer wearables such as Go-Pro cameras, smart glasses, and cameras embedded in household appliances, this trend is clearly on the rise. As a result, photo apps have become one of the most sought-after app categories for regular users. For example, 5 of the top 10 installed free apps in Google Play are related to photos [1]

Today, photo management requires capabilities beyond simple photo capture and storage. At the very least, photo management apps organize photos based on basic parameters like time and location of acquisition. Advanced apps have the ability to organize and search images based on complex image processing techniques such as face detection and object recognition. Such apps typically also allow their users to set specific tags for easy classification and retrieval of photos. Moreover, many photo apps have the ability to share photos across devices and users, and in doing so, they allow their users to manipulate their photos in various ways. For example, Instagram and Snapchat provide many image filters.

A typical photo management app manages the photos and the metadata separately. While the photo itself is built around the file abstraction where a single photo is represented by a file, the metadata containing the photo versioning, tags, search indices, organization etc., are decoupled from the photo and maintained separately—often an app-specific database. This lack of a unified interface to manage the photos entails three problems: First, it often leads to duplicate images resulting from filtering and resizing operations on the same photo by multiple apps. Second, the rich semantic information acquired from complex image processing cannot be shared among apps unless the apps are developed in tight cooperation. Third, when a photo is shared across multiple devices, it is hard to reason about propagating edit operations from one device to another when the images are adapted to each device.

Recognizing these problems, we argue that a more natural way of managing a photo is by encapsulating it within a logical object which has a set of attributes—for example, resized versions of the same photo, location where the photo was taken (location tag), object and face tags, etc. In addition, we believe that any app should be able to modify and access these attributes using a unified abstraction. Therefore, we propose to design a platform service for photo management—*Pixelsior*. Taking an example of Android, we envision this new abstraction to be provided in addition to, or as an advanced version of `MediaStore` [1], which currently provides metadata and URI for all available media data on a device.

Pixelsior treats a photo as an abstraction consisting of an original image, various derived images, associ-

[1] As of August 2015.

ated metadata, and version history. Given the increasing modalities through which photos can be generated and consumed (smartphones, tablets, laptops, televisions, smart watches and others), Pixelsior also adapts photos to suit the device by adjusting the resolution and other image properties.

This abstraction would ease app developers from the task of implementing photo management in the apps and instead allow app developers to focus on innovative features that differentiate their apps from other apps. By separating this into a platform service, such functionality can be availed uniformly by all apps instead of being implemented separately inside each one of them.

Our goal in this paper is to identify the design requirements for Pixelsior and discuss potential design choices. The contributions of our work are as follows—(i) we motivate the need for new abstraction for the management of photos through a platform service, and (ii) we explore the design space for the abstraction layer as well as for the service itself.

2 Motivation

In this section, we identify key challenges in developing photo apps for mobile devices to motivate the need for a new platform service. Three popular app categories, namely photo organizers, messaging, and social media platform, are analyzed. First, we briefly describe essential features in mobile photo apps. Then, we present the problems arising in implementing those features.

2.1 App Features

Popular photo apps provide features in three categories: *capture and store*, *retrieval*, and *image editing*. Each category can involve many extra operations as we discuss below.

Capture and storage: The most basic feature of photo apps is to capture a photo. Often this also includes capturing a set of metadata (such as location, date, etc.), and embedding within the photo. This metadata can be used later for photo apps in various ways.

Once an app captures a photo, the app stores the photo on its local storage. Additionally, the app might synchronize the photo with a cloud storage service and/or other devices; the app might re-size and compress the photo for faster viewing and sharing, and store the smaller image along with the original; It might also want to store additional metadata generated from computer vision techniques such as face detection, object recognition, and landmark detection.

Retrieval: Another basic feature of photo apps is retrieval. For retrieval and search, apps could allow photos to be manually organized into albums. Sophisticated apps use computer vision techniques to automatically organize and sort the photos by various parameters such as place, people etc. For example, if the user searches for "Niagara falls", the app would automatically retrieve all the photos taken in Niagara falls.

Image editing: Beyond storage and retrieval, many popular photo apps such as Instagram and Snapchat are built around image editing functionality. Edit operations vary in complexity from resizing to image filters, and as complex as manual retouching at a pixel level granularity. Several social interactions fueled by these features demonstrate their importance [2].

2.2 Challenges

The landscape today is a potpourri of different ecosystems providing their own (often partial) solutions to the three desired operations listed above. Technical and economic incentives have motivated companies to silo these operations in their own worlds, making it harder for users to have a uniform experience across apps and easily view, share, and manipulate images easily and consistently across apps. In abstract, here are some requirements to enable such functionality.

Unified view and interface: One of the major challenges in providing a unified view of the photos across the apps is the management of duplicate images that various apps create independently. Consider a specific case of mobile messaging apps which often reduce the size of images before sending them. If a user sends a photo to her friends and each of them uses a different messaging app, then the apps generate and possibly store multiple resized images of the same photo. A similar situation could happen in using social media apps. Posting a photo to a social media site after applying filters also results in an original image and its derived image. These duplicates are not desirable, result in excess usage of storage and bandwidth, and are annoying to users [2]. It is desirable to have a consistent view of photos across apps.

Metadata management: The second challenge is metadata management of the photos. As discussed, photos are tagged with metadata when they are captured. Moreover, many apps use image processing and machine learning techniques to associate each photo with rich semantic information. These trends result in a nontrivial amount of metadata to be managed.

Even though we have a de-facto standard format to store metadata in an image file (`http://www.exif.org/`), it is not flexible enough to support all kinds of metadata an app would like to use. Thus, an app builds its own custom metadata management tightly coupled with the app. A typical approach is to have a separate metadata database table where each relation includes metadata and links to image files. However, such an im-

Type	Interface	In	Out	Description
Read/Write	readPhoto	id, policy	image	Returns an image with the requested policy. A policy can ask to retrieve a specific resolution of the image or a more generic request like the "best quality", "fastest fetch", etc.
	writePhoto	id, image	none	Creates a new photo in the service.
Search and Query	search	list: query	list: photo	Returns a list of photos matching with a given list of queries
	queryPhoto	id	metadata	Returns metadata associated with the photo.
	queryPhotoHistory	id	history	Returns the photo's change history.
Transformation	transform	id, transformation	id	Applies the specified transformation and generates a new photo by recording the transformation. The transformation can be a built-in one of Pixelsior or app-specific.

Table 1: Basic APIs provided in Pixelsior

plementation does not allow multiple apps to share the metadata database unless the apps are developed in coordination. Therefore, any sharing of images results in two problems. First, it results in duplication of metadata across apps. Second, this makes the metadata management harder. Whenever a photo is updated or edited, the developer has to automatically update all the related metadata along with the photo. Failing which might leave the metadata in an inconsistent state. It is ideal to have no duplicates in both images and the image metadata for consistency.

Update management: The last challenge we discuss is management and propagation of edits to photos (like filters) across devices, often across different resolutions of the same image. Manipulating photos in mobile devices has become common and popular [3]. Assuming that a photo on a mobile device (the original image) is synchronized with another device, users would want the manipulation on the original to be propagated to the other device. This does not happen across apps in the current scenario unless the two apps have a tight coordination. It is desirable to be able to perform the same image manipulations on different apps.

In Section 3, we explore the design space for addressing these challenges.

3 Design Space

This section overviews the design space for our envisioned platform service for photo management. We separate our discussion into three parts—a developer's point of view, the internal photo representation, and update propagation. The discussion from the viewpoint of developers explains that Pixelsior is a platform service for apps. The discussion of our internal photo representation answers the question of *what* information needs to be managed in our platform service. The discussion of our implementation considerations examines the question of *how* the information can be managed. Our purpose is to discuss the factors we need to consider in designing a platform service for photo management ; it is not necessarily to present the final design. We leave the final design as our future work.

3.1 A Developer's Point of View

Table 1 shows the basic APIs that our Pixelsior design provides, which are available as platform service APIs to all apps running on a mobile device. The APIs have three categories—read/write, search, and transform. First, `readPhoto` interface allows an app to retrieve a photo with a set of policies. A policy can specify a resolution which an app will retrieve, e.g., a thumbnail of a particular size to display. It can also be used for automatic content adaptation done by Pixelsior such as "best fit for display" which triggers Pixelsior to determine the best resolution for display. Automatic adaptation can be done based on various contexts, e.g., display resolution and size, network connectivity, and battery status. `writePhoto` is a simple write operation that allows an app to store a photo.

Second, `search` allows an app to search and organize photos based on query strings on tags. A tag is a string attached to a photo, defined by an app, and can be used to group multiple photos into a collection; some example tags include locations, time and date, detected faces, etc. An app can use multiple tags for one photo.

Third, `transform` allows an app to edit a photo in various ways. `transform` takes a photo object and applies a photo edit operation. We envision Pixelsior would provide some built-in edit operations as well as an interface for an app to register its own implementation of an edit operation. This way, our design could allow the ease of development for apps that do not require advanced photo edit functionality, without preventing app develop-

```
{
  "images":[
    "orignal": {
      "uri":"https://url_to_image_object"
    }
    "thumbnail": {
      "derived_by": transform("original", thumbnail()),
      "content_file": "/local/path/to/thumbnail_file"
    }
    "1920x1080": {
      "derived_by": transform("original", resize((1920, 1080))),
      "content_file": "/local/path/to/file"
    }
  ]
  "tags":[
    "selfies", "loc: Buffalo, NY"
  ]
}
```

Figure 1: A proposed design of photo object internal representation in JSON-like format.

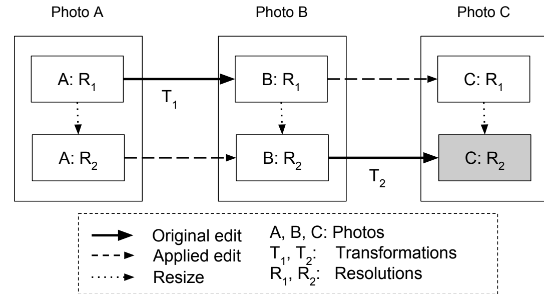

Figure 2: A sequence of photo edit operations. It shows an original photo (A) and its two resolutions (R_1 and R_2, as well as other photos (B) and (C) transformed in sequence. T_1 and T_2 indicate two different transformation operations and the solid arrows indicate which resolutions are used to transform.

ers from innovating in the space of photo editing. When an app applies an edit operation on a photo, the app can choose to either create a new photo object or include the result in the photo as a derived image. In either case, Pixelsior records the applied operations in the logical photo object. This allows apps to be *provenance-aware*. Such provenance-awareness allows an application to trace the transformation history of a photo back and forth, so the application can support a large set of operations such as undo/redo of photo edits, restoration of a deleted photo, quick retrieval of all transformed photos from an original photo, etc [4].

3.2 Internal Photo Representation

Pixelsior internally uses a new representation model to manage photos. This includes typical information for a photo, i.e., the name, the raw data, and the metadata (e.g., the file format). In addition, Pixelsior associates each photo with all additional metadata mentioned earlier in Section 3.1, such as tags and transformation history. An example of the proposed representation format is presented in Figure 1. Each photo object can include multiple images and each image can be stored either in local storage or in the cloud (specified as `uri` in Figure 1). In addition, each image is annotated with how the image is derived. In order to capture these different types of information, Pixelsior internally organize a photo object as a tree. Each photo object can have multiple versions of images and each image is represented as a node in the tree. If a photo is derived from another photo as the result of applying a manipulation operation, they form a parent-child relationship. A new photo just taken creates a new tree, while a transformed photo from another one either extends a tree or creates a new tree.

The actual management of these trees, as well as the generation of transformed photos, should be done efficiently by Pixelsior. In doing so, Pixelsior should take multiple front-ends into consideration (such as phones and tablets). For example, a phone front-end for a user might never retrieve a photo of the original resolution due to its small screen size or resolution. If this is the case, it makes sense for the phone front-end to only store scaled-down photos on the device. However, if the user edits a photo on another device, this update has to be propagated to the phone front-end. Now this update propagation can be done in many ways, and the service needs to implement an efficient mechanism. We discuss these implementation considerations next.

3.3 Update Propagation

We note that edited photos can be propagated to another device in one of two ways—either sending the result of edits directly or sending only the edit operations and applying them locally. However, always preferring one way over another will not yield satisfactory results. This is because the cost of sending edited photos can be unnecessarily high; on the other hand, always locally applying edit operations, especially a sequence of operations, might result in too much divergence as explained by Phan et al [8]. Thus, the service implementation should consider these factors and decide on what the best way is.

Figure 2 illustrates this further. The sequence starts with two versions of a photo with different resolutions, R_1 and R_2. For the sake of the illustration, it assumes that a device only access a particular resolution, e.g., the high resolution, R_1, on a laptop, the low resolution, R_2, on a mobile phone. In this case, if a user edits a photo on a device, it should be propagated over different resolutions. In Figure 2, solid arrows indicate the edits done by a user over different resolutions.

When propagating an edited photo across different devices, the platform may take the highest resolution copy, apply the edit, generate multiple resolutions, and distribute those to appropriate devices. It is likely that this will produce propagated photos with high quality. However, this is not the only way; for example, it is also possible to take a lower resolution copy and apply the edit operations. For example, if a device has the low resolution of Photo A and needs to generate Photo B, then the device may apply the edit operations to transform its copy of Photo A into Photo B. This possibility is indicated by dotted arrows in Figure 2 between Photo A and Photo B. Thus, the service needs to carefully evaluate various options and choose the most effective one based on contexts.

4 Related Work

Given the popularity of photos, a lot of work has been done in the recent years which deals with efficiently managing the photos, specially on the cloud. There is also some previous work on content adaptation in mobile computing. However we are not aware of any work that provides a unified abstraction for photo management and content adaptation in mobile apps. In this section, we discuss the related work in the areas of mobile data management and content adaptation.

Mobile data management: Simba [6] proposes a data-sync service for mobile and cloud apps with a unified interface for objects and tables. While Pixelsior also suggests a unified interface for accessing photos, our focus is mainly on simplifying photo management. Similar to Pixelsior, Earp [12] proposes a application-level data abstraction for mobile apps, but their focus is on fine-grained sharing and protection mechanisms.

Content adaptation: There is a line of previous work on supporting updates on adapted, low fidelity items and their reconciliation with full versions, in a scenario with PowerPoint and e-mail messages with media attachments [7], textual document [11], XML-based document formats [9], and images [8]. quFile [10] provides an abstraction layer that encapsulates different representation of the same logical data. The particular representation returned by quFile is determined at the time, depending on context and policy. Compared to quFile, Pixelsior supports editing operations on adapted data items and does not depend on the traditional file abstraction.

5 Summary

In this paper, we have motivated the need for a platform service for photo data management and a high level abstraction for photos. As a platform service, our design provides a unified view of photos to all apps and eliminates the complexity of metadata and photo editing management. Our abstraction keeps track of the changes of the photo, i.e., the original photo and how other photos are produced using image edit operations. This abstraction can be used to determine the best way to propagate updates across different devices. Our next steps are to implement the design of Pixelsior and demonstrate its utility.

Acknowledgements. This work has been supported in part by an NSF CAREER award, CNS-1350883. We thank anonymous reviewers for their helpful comments.

References

[1] MediaStore. `http://developer.android.com/reference/android/provider/MediaStore.html`. Accessed: 2016-05-15.

[2] My little sister taught me how to "snapchat like the teens". `http://goo.gl/wipU5C`. Accessed: 2016-03-10.

[3] BAKHSHI, S., SHAMMA, D., KENNEDY, L., AND GILBERT, E. Why we filter our photos and how it impacts engagement. In *9th International AAAI Conference on Web and Social Media* (2015).

[4] CARATA, L., AKOUSH, S., BALAKRISHNAN, N., BYTHEWAY, T., SOHAN, R., SELTZER, M., AND HOPPER, A. A primer on provenance. *Commun. ACM 57*, 5 (May 2014), 52–60.

[5] EVANS, B. The explosion of imaging. `http://goo.gl/0AdkzQ`. Accessed: 2015-08-13.

[6] GO, Y., AGRAWAL, N., ARANYA, A., AND UNGUREANU, C. Reliable, consistent, and efficient data sync for mobile apps. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2015), FAST'15, USENIX Association, pp. 359–372.

[7] LARA, E. D., KUMAR, R., WALLACH, D. S., AND ZWAENEPOEL, W. Collaboration and multimedia authoring on mobile devices. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services* (New York, NY, USA, 2003), MobiSys '03, ACM, pp. 287–301.

[8] PHAN, T., ZORPAS, G., AND BAGRODIA, R. Middleware support for reconciling client updates and data transcoding. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2004), MobiSys '04, ACM, pp. 139–152.

[9] PUTTASWAMY, K. P., MARSHALL, C. C., RAMASUBRAMANIAN, V., STUEDI, P., TERRY, D. B., AND WOBBER, T. Docx2go: Collaborative editing of fidelity reduced documents on mobile devices. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2010), MobiSys '10, ACM, pp. 345–356.

[10] VEERARAGHAVAN, K., FLINN, J., NIGHTINGALE, E. B., AND NOBLE, B. qufiles: The right file at the right time. *Trans. Storage 6*, 3 (Sept. 2010), 12:1–12:28.

[11] VEERARAGHAVAN, K., RAMASUBRAMANIAN, V., RODEHEFFER, T. L., TERRY, D. B., AND WOBBER, T. Fidelity-aware replication for mobile devices. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services* (New York, NY, USA, 2009), MobiSys '09, ACM, pp. 83–94.

[12] XU, Y., HUNT, T., KWON, Y., GEORGIEV, M., SHMATIKOV, V., AND WITCHEL, E. Earp: Principled storage, sharing, and protection for mobile apps. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, Mar. 2016), USENIX Association, pp. 627–642.