Copernicus: Face-to-Face Web-based Sharing and Collaboration

Jing Su, David Dearman, Dritan Xhabija
Dept. of Computer Science
University of Toronto

Ashvin Goel
Dept. of Electrical and Computer Engineering
University of Toronto

Eyal de Lara
Dept. of Computer Science
University of Toronto

ABSTRACT

Mobile devices are increasingly being used to access web applications. Unfortunately, the usage model for web applications today is still desktop-centric, in which users work in isolation. As a result, using mobile devices to find or share data, or collaborate while in a social setting is cumbersome at best, and often frustrating.

Instead, we envision a new usage model in which web applications supplement and enhance face-to-face social interactions. This model forms the basis of Copernicus, a system that simplifies the sharing of web content between individuals that come within proximity of each other. Copernicus introduces a collaboration model in which users in proximity are represented as first class objects in the system, which encapsulate the users' web applications and content. Copernicus allows building applications that can take advantage of user objects to enable sharing and collaboration with nearby users.

1. INTRODUCTION

People are increasingly making their personal content available on the Internet by hosting it on a variety of web applications. Extremely popular web applications include FlickR for photos, Google Calendar for scheduling, and Imeem for sharing music. This has resulted in a situation where users have two modes of interaction: traditional face-to-face interaction in the real world, and computer-based asynchronous interaction on the web (e.g., commenting on a friend's photographs). Our aim is to enable people to socialize simultaneously in both modes by providing seamless access to the user's web content during face-to-face interaction.

Consider the current situation. Say Bob is a supplier who is visiting a group of people at a prospective client company. Bob has a portfolio of web hosted content, including documents, videos, CAD files, and other media, that he wishes to share with his clients. Some of this content is sensitive, hosted on Bob's company's website, and requires authenticated access. Yet other less sensitive promotional content is hosted on third-party web applications, such as a media streaming service, where for example, a "list of friends" allows accessing content. Providing access to these relevant materials is not a simple matter of emailing links and attachments; logins and usernames must be determined or created, access permissions granted, and directions for where and how to access this content provided. This complex set of operations are cumbersome and disruptive during a face-to-face meeting, especially if this is the first introduction between Bob and many of the clients. To mitigate this problem, Bob may bring in mass-storage media with the content pre-loaded. However, now the onus is on the clients to load the media and ensure they have the proper software to view it. This approach is still problematic if the clients primarily use smartphones.

Continuing the previous example, suppose that Bob decides to setup a follow-up meeting with the clients. Though this collaborative task is conceptually simple, it involves users focusing on their calendars on their mobile devices, followed by several rounds of verbal interaction in which people suggest their free times and others who respond with their busy times, and then all users updating their calendars. These examples illustrate that while sharing content and collaborating in social settings is desirable, it is cumbersome and rarely done. Instead, these tasks are done later, and may involve multiple rounds of emails for coordination.

In this paper, we present Copernicus, a mobile system that lets people socialize at both the physical level and on the web by providing a bridge between the two worlds. Copernicus introduces a collaboration model in which users in proximity are represented as first class objects in the system. These *user objects* encapsulate the users' web applications and content. This abstraction provides users with an intuitive model for finding and sharing content and collaborating with nearby users. The result is a computing experience in which digital content is seamlessly made available to users interacting in the real world, instead of forcing users to focus on their mobile devices and retreat away from the social interaction in order to navigate the web.

Revisiting our example, Bob uses his Copernicus-enabled mobile device to access the web content that is to be shared.

University of Toronto Technical Report CSRG-603

He then selects a Copernicus specific "share" option from the menu, and is presented with a pop-up of thumbnails of the people around him. Bob selects the clients, and confirms the sharing operation. The clients find a dynamically generated directory containing the files that Bob has shared on their own favorite documents web site (e.g., an internal web site or Google documents). Note that Bob does not require the clients' email addresses, and the clients do not have to login to Bob's web sites to access the shared documents. In fact, they may not even be aware of where the files are actually hosted. Similarly, the participants set up the next meeting time by opening their calendar on their mobile devices and sharing all their free times with others in the room. Then, Bob sees a highlighted entry two weeks from now in his calendar, indicating that everyone is free on that date and time. Bob selects that entry, books the time, and the clients' calendars are all updated with that entry. Bob shakes hands and looks forward to the next meeting.

Copernicus makes two main contributions that enable these collaborative interactions. First, it provides support for user objects that represent users in proximity. Copernicus detects individuals in the proximity of a user by mapping low-level device identifiers such as Bluetooth MAC addresses of mobile devices to real individuals, and then mapping individuals to their web applications, login ids and their content. The user objects thus encapsulate a user's applications and content. This approach allows building new web applications that can take advantage of these objects for seamless sharing and collaboration. Second, Copernicus provides an adaptation service that retrofits existing web applications to enable sharing and collaboration with nearby users. This service takes advantage of the programmable interfaces available in modern web applications.

We have built and evaluated a prototype of the Copernicus system using smartphones. Our prototype includes support for several applications (calendar, email, photo and file sharing) from different web providers, thus showing the generality and wide applicability of our approach.

2. APPROACH

Copernicus simplifies web-based sharing and collaboration during face-to-face social interactions by providing support for user objects that represent users in proximity. User objects have a human friendly representation, as shown by the list of faceshots in Figure 1(b).

Alice can share some of her content by simply selecting her content, clicking on the share button as shown in Figure 1(a) and choosing the faceshot of the receiving peer. Conversely, Bob can easily find content that has been shared with him, as shown by the links in Figure 1(c). Bob can also filter content by peer by using a faceshot selection interface. Under the covers, Copernicus takes care of the intricacies of identifying users in proximity, sourcing content from the origin web applications, enforcing access control, converting between data types supported by different web applications (e.g., Picasa and Facebook), and presenting the shared content in an intuitive manner.

The rest of this section describes the two main ideas, user objects and an adaptation service, that Copernicus uses to support face-to-face web sharing. We then describe the range of sharing capabilities and applications enabled by Copernicus. Finally, we address security and privacy concerns.

2.1 User Objects

User objects represent users in proximity and they encapsulate a user's web applications and web content. Copernicus provides this abstraction by using three mechanisms that are discussed below.

Identify users in proximity

Copernicus needs to detect individuals in the proximity of the user. It can leverage different techniques, such as a GPS-based location service, to inquire about users within a specific geographical area. Our existing prototype leverages short range radio (e.g., WiFi, Bluetooth) to detect the MAC addresses of other mobile devices in close proximity. Short-range wireless radios are available on most smartphones today and they provide good accuracy for determining physical proximity. Next, Copernicus maps the low-level hardware identifiers to user objects. This mapping is provided by users when they initially register with the system. The eagerness with which Copernicus updates the user's proximity depends on user preferences and applications, as discussed later. Copernicus can also keep a log of users that have come within proximity of the user for later use (e.g., to simplify sharing with someone we met earlier in the day).

Determine the user's web applications

Copernicus is aware of each users' web applications. This mapping is also provided to Copernicus by users when they register with Copernicus, and it allows linking a user object with a user's applications.

Enable access to web content

The last aspect of a user object is enabling access to the user's web content. Copernicus aims to provide seamless access to web content, across different web applications and with different user memberships. Meeting this goal places two requirements: access control and content interoperability.

Copernicus accesses user content using delegated credentials, a widely available feature of Web2.0 applications for enabling third-party access. Copernicus is provided with these credentials (unique tokens) when users register their web application. Copernicus can then make requests to access the application content on behalf of the user. Requests are signed with public/private key certificates and the web application checks the IP address of the request to ensure that it originates from Copernicus. We discuss the security concerns with credential delegation later in this section.

When users share content, Copernicus attempts to use the native access control settings of a web application whenever possible. However, these settings may be insufficient

(a) Share menu option (b) Peer selection (c) Finding content

1: *Sharing and finding content. Alice initiates content sharing by using the share menu option or by using an embedded link tagged with the content. A peer can be selected easily by clicking on a faceshot (or by typing in a cellular number). Bob can find shared content by clicking the different links, each of which implements a different filter.*

due to user privacy settings, insufficient granularity in the application's access control mechanism, or lack of common service membership between users (e.g., the peers do not use the same web application). In this case, Copernicus securely proxies content using the delegated credentials of the content owner, thus enabling access control at the user object granularity. For example, Alice can share a specific document with Bob, even though Bob does not have an account with the service used by Alice.

Seamless content access also requires content interoperability between different web sites. Copernicus provides a framework and API for translating content into a per-application class intermediate formats. It is important to clarify that Copernicus is not a content repository. Instead, its API enables content to be linked and shared across web applications.

2.2 Proximity-aware Adaptation Service

The user object abstraction described above allows building new web applications that provide seamless sharing of web content and collaboration during face-to-face interactions. However, we would also like to retrofit existing web applications to achieve the same goals. Our solution is to provide a service that utilizes the APIs provided by modern web applications to adapt these applications.

For example, the Copernicus adaptation service allows filtering, searching and sorting of web content. As the amount of user-generated and shared web content increases, these operations become burdensome in a mobile setting, even though they may be as simple as marking a piece of content to be shared, or vice versa, finding a piece of marked content.

The adaptation service adapts the organization and layout of content based on the user's current and past proximity context. We apply this approach to both the user's own content as well as shared content. In the introductory example, when Bob opens his calendar to arrange a follow-up meeting with his clients, the adaptation service inserts temporary "free" entries that match the intersection of available times of all participants.

The adaptation service provides two interfaces. It can serve as a web portal that provides access to the adapted content. This Copernicus portal presents a series of links containing the adapted content from both the user as well as others. For example, when Alice visits the portal, there is a link that shows the content she has shared with the peers in her proximity, a link that shows the content of other people in nearby proximity, and a link that shows the content that has been shared with or received by Alice. These links provide Alice with a convenient and fast way to find and see the content that Bob has shared with her. For example, Alice can access Bob's shared photos and videos hosted on Facebook and YouTube, without even knowing the web applications that Bob uses.

The Copernicus adaptation service provides a second, more intuitive interface by directly adapting the web application. In this case, users directly interact with their application rather than visiting the Copernicus portal. This approach has the added advantage that it leverages the efforts of web application providers in creating custom clients and interfaces for mobile devices. For example, say Bob has taken several photos at a conference and shared them with Alice. When Alice visits her Picasaweb, she finds a virtual album inserted by the Copernicus adaptation service containing thumbnails of the shared photos. Opening the thumbnail photo provides Alice with a link to the full-quality photo, with permission settings automatically set up by Copernicus. Figure 4(b) provides a screen-shot of how Bob's photos on Facebook can be seen by Alice, despite Alice not having a Facebook account.

2.3 Sharing Policies

Copernicus enables a range of sharing models, based on the person sharing the content (*sender*) and the user which whom the content is shared (*receiver*). Applications and

tasks may require explicit interaction from both the sender and receiver, or just sender or receiver, or be completely automated without requiring any user interaction.

Copernicus provides an intuitive point of contact for initiating the sharing of content. However, we note that users are able to share content with others who are not in their physical proximity, for example people they have met previously. Whether shares persist is application specific; some applications may enforce share permissions to be proximity-based, while others may allow shares to persist until revoked by users.

2.3.1 Explicit sender/receiver interaction

In this model, the sender explicitly marks the content to share and the individuals with whom to share (e.g., by clicking on their picture on her mobile device). Similarly, the receiver explicitly chooses who to receive content from. Many applications fit into this model: sharing of documents, photos, or music, and scheduling meetings.

To help filter friends from strangers when sharing content in a crowded space, Copernicus can utilize relationships available in social networking applications. In our example, suppose Bob wanted to share some of the photos he took with Alice. When he initiates the sharing operation, Copernicus presents a list of individuals with whom he can share content. Eve, a stranger, is nearby but does not show up on the list because she is not in Bob's social network.

Though this explicit model requires user interaction, it is less taxing for the user than existing approaches. Instead of working with device identifiers and machine names, users are able to interact much more naturally by specifying their peers as people instead of arbitrary IDs.

2.3.2 Implicit sender or receiver

An example of an application that provides a mixed mode of operation, where user interaction is required from only the sender or receiver, is a business card exchange web application. Unlike existing business card formats such as vCards, which are restricted in content and format for size purposes, a business card application on Copernicus can easily exchange arbitrarily large and rich portfolios since the sharing is performed by well-provisioned servers. The client devices (in this case smart-phones) do no work.

Imagine a pink-slip party, where individuals who have recently lost their jobs can network and commiserate, and recruiters can look for potential employee candidates. In this setting, a user looking for work can set his portfolio to be available automatically to anyone interested. Conversely, a recruiter may wish to be more selective with which portfolios to accept. In this usage mode, the sender is offering content without requiring user interaction. However, the receiver chooses to selectively and interactively accept content. At the same time, potential employers may also be proactively recruiting well-qualified candidates, offering them portfolios highlighting the benefits of working at their company. The candidates happily accept all such solicitations from potential employers. In this usage mode, the sender selectively decides who to send content to, but receivers automatically accept the shared content without user interaction.

2.3.3 Implicit sender and receiver

On the opposite end of the spectrum is a usage model where the application implicitly shares and accepts content on behalf of users. There are many possible applications which may opt for such a usage model. For example imagine an application that enables Alice to take a tour of a city, without taking any photos of her own, and then later be able to build a photo-album from the public photos of the people she took the tour with or even random strangers who happened to be in proximity.

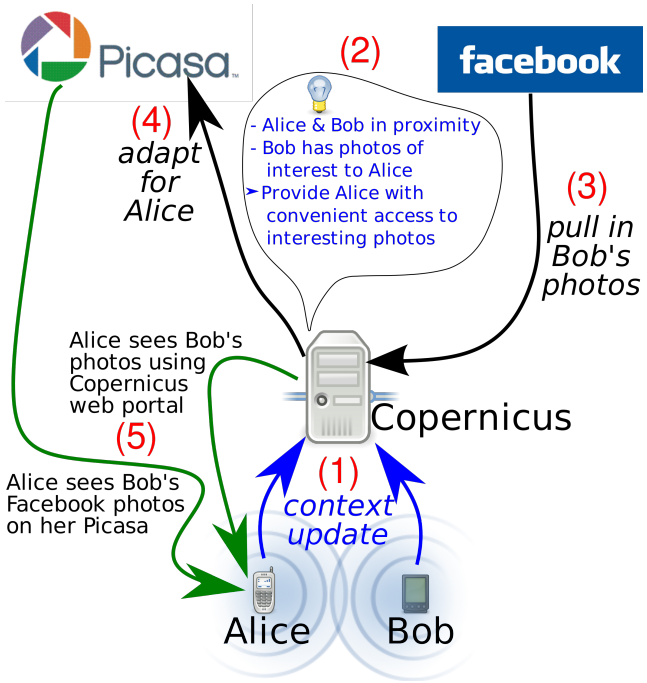
2.4 Security and Privacy

Copernicus simplifies sharing by providing a list of peers with human-identifiable attributes (such as real name and profile photo), and mapping their online identities, accounts, and web applications. This new usage model raises security and privacy concerns as well as the risk of receiving unwanted content such as spam.

The current Copernicus system relies on periodic Bluetooth inquiries for detecting user proximity, which raises spoofing and tracking risks. First, while most consumer products with Bluetooth radios are hard-wired with a Bluetooth MAC address, a malicious party may attempt to build custom hardware that transmits fake Bluetooth address responses, allowing impersonation. Such an attack would have limited effectiveness in Copernicus, since it does not allow the attacker to gain access to the content of the impersonated identity. For example, suppose Alice wants to share with Bob, but Eve impersonates Bob's Bluetooth address. Alice would be tricked into thinking that Bob is nearby. However, since Copernicus sets access permissions on the originating web application, Eve can only gain access to the content by compromising Bob or Alice's accounts on the web applications.

Second, Copernicus does not expose users to user tracking risks above the current status quo for identifying mobile devices. Existing methods have explored techniques for overcoming this social presence problem [6], which we leave for future work. In addition, users can disable Bluetooth by default, and enable it explicitly when they want to share. To mitigate the risk of Bluetooth inquiry response tracking, our prototype provides a method for direct peer specification via a cellular phone number. This enables the recipient to remain completely invisible, while being able to receive sharing requests. We describe the use of SMS, as well as other possible techniques for establishing proximity without the use of active Bluetooth inquiries, in the architecture section.

Malicious users may attempt to spam other users in their proximity by sharing unwanted data with them. Copernicus enables users to restrict their visibility to individuals with whom they have previous sharing relationships or are marked as acquaintances on a social networking application.



2: *The Copernicus Service*

In addition, Copernicus provides users with filtering and rejection options analogous to email spam filtering.

The use of delegated credentials is common amongst web applications, and can take many forms, ranging from "home-brew" delegated credentials to delegation through single-sign-on services. In all cases, Copernicus does not store passwords. Access to the delegation token does not grant an attacker access to the user's web application data, since web application requests must be signed from a specific IP address associated with the token. An attacker must compromise the Copernicus portal, or steal its private key and spoof its IP address to gain access to the user's data. Thus, Copernicus's security profile is similar to other web applications using credential delegation, and Copernicus does not expose additional vulnerabilities. In particular, users who wish to unregister from Copernicus can revoke the delegated credential from their web applications.

3. ARCHITECTURE

Figure 2 shows the main components of the Copernicus service: the Copernicus client running on the users' mobile devices, the Copernicus service, and various third-party web applications, such as Facebook and Picasaweb. The example illustrates how Alice can access photos that Bob has shared with her. These photos are available to Alice, without her needing to know that Bob uses Facebook, the albums in which his photos are stored, or requiring membership to Facebook to gain access.

3.1 Copernicus Mobile Client

The Copernicus mobile client consists of a web browser plugin that adds a share menu to the browser and scans the local radio environment to help identify users in proximity. The current implementation scans for nearby Bluetooth addresses and collects cellular and WiFi radio fingerprints. The device sends the results to the Copernicus service via a secured connection through an infrastructure-based Internet link, e.g., WiFi or 3G. This client is necessary because acquiring this information requires device-specific code.

3.1.1 Sharing

Copernicus provides three methods for initiating a sharing request. When supported by the application, users can select a *share* link embedded within the application content. Alternatively, users can install the Copernicus browser plugin and click on the *share* menu item it adds to the browser menu (Figure 1(a)). Finally, as a last resort, users can select a *share* link from Copernicus's web portal.

Once Copernicus has determined that sharing is allowed, the user's browser is redirected to a peer selection page hosted by Copernicus, shown in Figure 1(b). Here, the user is presented with a list of individuals with whom to share content. There is also an input field for the user to manually enter a cellular number, in case the peer is not a registered Copernicus user. Once a peer has been selected, Copernicus sets up the appropriate access permission settings and redirects the user back to the originating web application.

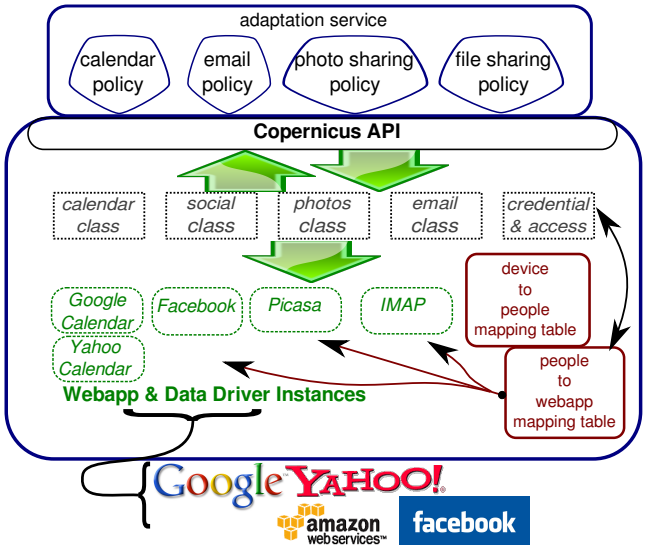
3.1.2 Bootstrapping and notification

Copernicus utilizes SMS notification for two functions: explicit user identification and bootstrapping of new users. We chose to use SMS because it is a well supported feature on cellular devices and is operated by trusted carriers. Furthermore, exchanging cellular phone numbers is a socially accepted practice amongst acquainted individuals.

For a user wishing to remain "invisible" as well as users who are not registered with Copernicus, a peer wishing to initiate sharing can do so by entering the user's cellular number in the recipient selection page described in the previous section. Copernicus then creates a new anonymous user account, establishes the sharing permissions for that account, and creates a unique URL hosted by Copernicus with which to access the shared content. This URL is then sent to the recipient's cellular number via SMS. The recipient is then able to access the shared content via the URL. The recipient user can merge this anonymous account into their own Copernicus account at a later time via the Copernicus portal. Similarly, a new user can register on Copernicus and integrate this anonymous account.

3.2 Copernicus Server

Figure 3 shows the architecture of the Copernicus server. The adaptation service implements application-specific policies and interfaces with the Copernicus API. This API covers a range of popular user applications, such as calendar and photo sharing, and specifies common operations and data types for these applications. Copernicus includes driver modules that interface directly with specific web applications, and implement the code to connect with the Copernicus API. For example, there is a Facebook



3: *The Copernicus architecture*

Photo driver and a Google Picasaweb driver, which both implement the Copernicus photo API. The Copernicus APIs define more than just data access functions. They also define operations for reorganizing content, managing dynamically created content, and useful hotlinks for initiating content sharing operations. How these operations are manifested depend on the specific web application driver implementation. In addition, the Copernicus server also contains a name mapping module that maps individual users to their devices and their online identities, a proximity detector, and a credential lending and access control module. The name mapping module exposes individuals as user objects via the Copernicus API.

3.2.1 Adaptation Service

The adaptation service implements application policies as Copernicus-native programs. These programs specify domain specific algorithms for how content should be filtered, sorted, organized, merged, and shared. Copernicus supports two primary modes for how these content management decisions are applied: a *now* mode, and a *past* mode. The *now* mode applies policies based on the user's current proximity context, as determined by the proximity detection module. The *past* mode enables the user to apply policies based on accumulated previous proximity context. For example, suppose Alice forgets to schedule a follow-up meeting with Bob. Alice can effectively "turn back the clock" to her meeting with Bob, and see the adapted policies as if Bob were in proximity.

3.2.2 Web application classes and drivers

The web application interfacing subsystem has two main components: a library of application domain classes and intermediary data formats, and web application-specific drivers which implement these interfaces.

Copernicus uses drivers to cope with the application-specific nature of the APIs exported by the different web applications. Drivers must implement one or more application class interfaces, which basically encompass four core functions: pull content, push content, manage access control policies, and insert user interface extensions. Due to the diversity of features and capabilities across web applications, drivers are free to determine how the content organized by Copernicus is to be applied. For example, to add dynamically created calendar entries, the Google Calendar plugin may create supplementary calendars for the entries, which can then be overlaid atop the user's primary calendar. A plugin for Yahoo! Calendar, which does not support secondary calendars, would add the new entries directly into the user's primary calendar.

To minimize the transfer of content between the web applications and Copernicus, a driver can implement an optional interface for allowing filter expressions to be pushed to the web application. This enables significant filtering and selection operations to be evaluated locally on the web application's system. Many web applications support some form of search or query interface, which can filter across different content types (e.g. FBQL [7]) or tag and location attributes (e.g. FlickR search [8]).

To support operating on data formats from many different web applications, Copernicus's application class interfaces define an intermediary format for every supported data type. Copernicus specifies intermediary formats for images, calendars, emails, social networking relationships, and meta-data such as geo-tags and radio signatures. This common set of formats enable the filtering system to work on content independent of their originating or or destination web applications.

3.2.3 Name mapping

Copernicus must solve two important questions: who is the individual that owns a given mobile device, and where are that individual's online identities and resources? To provide these two mappings, Copernicus's name mapping module assigns each individual a unique user account with which devices and online identities are associated. Human and socially identifiable attributes such as a name or profile photo can be directly specified by the user to Copernicus, or the information can be extracted from the user's other services such as Facebook. Copernicus's name mapping module assumes that devices, device identifiers, and online identities are owned by a single person, though a person may own many devices and many online identities.

3.2.4 Proximity detection

The proximity detection module detects nearby peers by processing the radio environment updates submitted by the Copernicus mobile client. The proximity module appends the received updates to a ring buffer, and supports multiple proximity detection modules, by allowing each of them to scan over the update buffer and raise notifications for proximity changes or updates. The proximity module then determines how to weigh the notifications from various modules, depending on the user's preferences. The

Copernicus architecture supports proximity modules based on a variety of sensors, such as GPS readings, WiFi and cellular fingerprints [2, 5, 21], and Bluetooth scanning. The framework can also accommodate approaches, such as Amigo [22], which detect proximity based on similarities in the radio signal fluctuations perceived by nearby devices.

3.2.5 Credential lending and access control

Copernicus uses credential delegation, a widely available feature on Web2.0 applications, to allow third parties to make requests on behalf of a user. This credential lending module is used for three purposes: to sort and filter content on behalf of the user, to retrieve content for purposes of sharing between users, and whenever possible, setting access permissions for content hosted on webapps.

The access control module in Copernicus determines which of the filtered content can be accessed by the receiver. By default, Copernicus extracts and utilizes the settings from the user's webapps. This provides the user with an already familiar access model. The exception is when users explicitly share content with specific receivers. This fine-grained explicit sharing over-rides the default rules.

In many cases web applications are unable to grant this fine-grained explicit sharing request due to two possible limitations: insufficient access control granularity, and the lack of mutual membership on the web application between sharer and receiver. Copernicus's access control module tackles both of these problems by serving as a secure content proxy. When providing access, receivers and the receiver's web application are given uniquely generated URLs hosted by Copernicus. Every time these unique URLs are accessed, the access control module first checks the identity and permissions of the accessing device. If permission is granted, Copernicus securely retrieves the shared content, and proxies access to the receiver.

4. IMPLEMENTATION

The prototype Copernicus system is implemented using the Apache Tomcat Java Servlet framework. For user devices, we implemented support for the Windows Mobile 5 smartphone platform. In addition, an initial implementation for the Apple iPhone platform is partially complete as of this writing. In the remainder of this section we provide a brief description of the implementation details of the prototype applications. We show the generality and wide applicability of Copernicus by implementing applications from different domains and web providers.

Calendar

The calendar application aims to simplify the task of finding common free times for establishing future meetings. This task can become especially time consuming as the number of participants increases and the complexities of schedule times and conflicts result in burdensome communication overhead.

We implement support for the Google Calendar service. The calendar application adaptation policy produces the set of intersecting free times. This set is given back to the plugin which pushes this result back into the users' Google Calendar (see Figure 4a). This implementation creates a temporary calendar which can be overlayed on top of the user's primary calendar. To mark these entries as Copernicus-generated, the description fields for the entries are tagged with a hash check-sum generated from the event details. This enables Copernicus to find and update these entries later when the user's social proximity changes. The description fields also contain links to common actions, such as reserve a meeting at this time slot. This is accomplished using a link to a uniquely-generated URL on Copernicus which triggers the action.

Photo Sharing

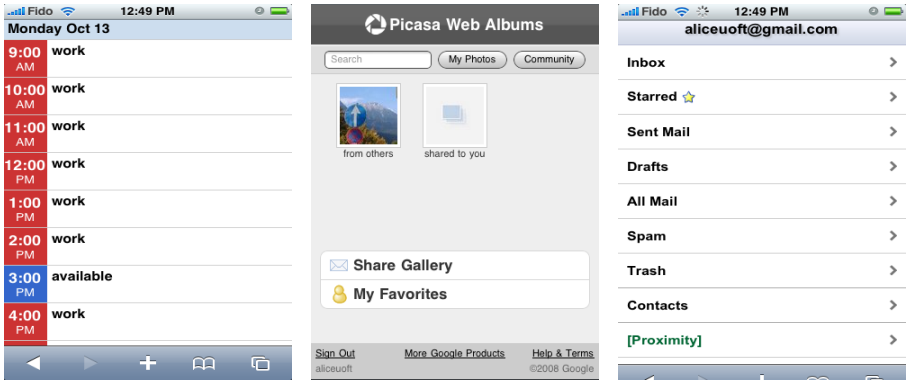
The photo sharing application aims to enable users to find and share photographs of interest between people in a social group. The Copernicus photo application automatically constructs virtual photo albums for sorting a user's own content, as well as displaying content shared to the user by others. This feature works across different web applications, enabling users to find both their own content as well as shared content from their preferred service without having to figure out where the peer's content is stored or how to access it. As the user's physical proximity changes, Copernicus automatically manages these virtual albums, adding or removing photos. An example of this can be seen in Figure 4(b), where Bob has shared a photo from his Facebook album with Alice. Alice can see and access the photo from her Picasaweb application, without needing Facebook membership.

Copernicus provides photo sharing functions through Picasaweb using the Google APIs, and Facebook photo albums using the Facebook API. The current photo application has two policies: one which selects the set of photos Bob took when Alice was in close proximity, and one which selects explicitly shared photos. These filter results are instantiated as virtual albums populated with thumbnails of the selected items, all managed by Copernicus. Appended to each thumbnail is an identifying hash-tag and full-image link, hosted by Copernicus which enables Alice to see the full sized original.

Email

The email application aims to make it simpler for users to quickly and easily find relevant related correspondence between the user and their peers in proximity. Existing solutions for finding email include manual sorting into folders or tags, using textual search, or sorting by date or sender. All three of these methods are less than smooth in a mobile environment. Typing a search term may be convenient on a desktop, but is clumsy on a mobile device. Finally, sorting by sender is often too coarse-grained. For example, people can belong to different project groups, all with their own threads of correspondence. Sorting only by a specific sender does not properly distinguish between different project threads, implicit in the list of included recipients.

The Copernicus email application creates a virtual folder



(a) Calendar (b) Photo (c) Email

4: *Copernicus applications. The Calendar applications shows the union of busy times. The Photo and Email applications show a dynamically "Proximity" folder or gallery with pictures shared by people in proximity or emails from people in proximity.*

sorted with email correspondence from the included participants in proximity, as illustrated in Figure 4c. Unlike simple sort-by-sender, Copernicus email can sort by all-senders-in-proximity, which is much more fine-grained and selective. As the user's physical context changes, the messages in the folder are dynamically updated.

Despite the widespread popularity of web-mail applications, there exists no Web2.0 APIs for manipulating and managing email. However, a basic set of adaptations can be performed using the IMAP protocol, which is almost universally supported by email services. In our implementation we use Google Mail service as our IMAP email back-end, though nothing in the IMAP plugin implementation's design is specific to Google Mail.

5. EVALUATION

In this section we evaluate the performance of our Copernicus prototype. The mobile device used is a JAQ3 Windows Mobile, manufactured by HTC, which is an EDGE capable GSM phone with WiFi, color touch-screen, and hardware QWERTY keyboard. The Copernicus server is a quad-core Xeon 3.6 GHz with 4 gigabytes of RAM, running Apache Tomcat.

For each applications introduced in the previous section, we profile the interaction requirements to achieve common tasks using Copernicus, and compare with alternative existing models for sharing content. Tasks are performed at least three times, showing the averages of the interaction counts and time-to-completion. All tests are performed by users with strong familiarity and proficiency with the applications and devices. The objective is to determine how an experienced user would utilize the system, since even novice users eventually become experienced.

Calendar

We evaluate the time and number of interactions required to schedule a meeting time which is mutually agreeable to two individuals that are in proximity. Both calendars are pre-loaded with busy schedules such that 3 PM a week from today is the first available free time. Participants are tasked with finding and scheduling this follow-up meeting. Each participant is first given time to acquaint themselves with the schedule given to them. Timing and counting of interactions begin at today's calendar date, and end when both participants have agreed on a meeting time.

The first row of Table 1 shows the time and effort it took for the users to arrange a meeting without Copernicus' support. This scenario requires users to look up their own calendars, and verbally communicate their free-time options. The second row of the table, labelled *Oracle Calendar* shows the time and effort required to set up the meeting in an idealized scenario where both users knew a priori exactly when they have a mutually free slot. The difference between these two rows shows the basic social human communication overhead to negotiate and find this time. Finally, the last row of the table shows results for Copernicus, which helps users by highlighting common free times. By having the right content adapted into the user's application, Copernicus enables participants to finish this task 70% faster than with the native application, and matches the performance of the idealized case.

A further benefit of Copernicus over current systems is that Copernicus maintains this simplified usage model even as the number of meeting participants increases. As the number of participants go up, so do the complexities of time constraints, and the odds of immediately knowing a definite common free time will be significantly reduced. Copernicus, in contrast, scales well to support larger social interactions.

Email

In this experiment, we examine a use case where Alice checks her email in order to reference a message Bob sent to her earlier. The purpose of this experiment is to show the potential for how automated sorting and filtering can be applied to a user's own content, and that such adaptations can be effectively applied using IMAP. All experiments use the GMail web interface.

To set up this experiment, we assume that Alice receives an average of 25 non-spam emails per day. The particular

correspondence from Bob she is looking for took place three days ago, with the subject line containing "Fact Check". When Alice and Bob are in close proximity, Copernicus automatically sorts and creates a "Proximity" folder sorted with correspondence with those who are currently nearby. For each of the trials, timings start with the email client open at the top of the inbox, and end when the desired message is selected.

Copernicus significantly outperforms native GMail requiring up to 78% fewer click and taking up to 65% less time to complete the task. We attribute native GMail's poorer performance to two reason. First, to filter emails by sender name or subject line using native GMail, the user has to type a search term. Second, the web interface only shows 25 mail messages per page, requiring extra navigation time to scroll to the bottom and find the "next page" link, and extra page load times to transition to the next page of messages.

Photo Sharing

We compare Copernicus to two usage models for sharing photos: sending a link to the content, and sending the content itself. In order to compare the sharing usage models, user interaction measurements begin when the sender's device has the desired photo loaded and ready. For Internet-related methods, this means the photo is loaded on the sender's browser, hosted by Picasaweb. For file transfer based methods, this means the photo loaded in the sender device's native photo viewer. Measurements stop when the receiving device has received and loaded the photo. Two sizes of photographs are tested: 36 kilobytes, typical of a low-end camera-phone; and 358 kilobytes, typical of a well-compressed medium-range camera-phone.

The first sharing model we explore involves sending a URL link of the photo of interest to the intended recipient, via email and SMS. Most popular email and SMS clients today are able to detect and hyperlink URLs in the text. The sharer addresses the link to "bobuoft" as the recipient address (the address book automatically fills in the remainder @gmail.com suffix). For SMS, the sharer enters the receiver's 10 digit cellular phone number. In the email experiment, the receiver must manually force a new mail check, instead of waiting for the usual 10 minute periodic mailbox check. A limitation of this sharing model is the complexity of the permission granting process if the sharer's photo is not at a publicly accessible URL. For this test, the photo to be shared is set as publicly accessible. The *link* rows of Table 1 summarize the sender and receiver interactions as well as end-to-end time for sharing the photo. Because only a link is sent, there is little difference between file sizes of the content.

The second sharing model involves transferring the actual photo file to the recipient device. We compare three methods of sending content: via email attachment, Bluetooth OBEX file transfer (BT), and MMS. We assume the sharer has a copy of the photo cached on her mobile device. The *attachment* rows of Table 1 summarize the result of this experiment. Unlike the link sharing model above, file sizes matter in these trials since binary file content must be transferred from one device to another. In the email attachment trials, there is a notable increase in the completion time compared to the email link sending trials, which is caused by the uploading and downloading of the image attachment. The difference in interactions required on the sender is simply due to differences in menu clicks on the device to initiate the email composition. For the Bluetooth trials, a large amount of user time is spent waiting for Bluetooth device discovery, followed by the transfer time, which can be slow on some devices. We note that photos larger than 250K in size could not be sent via MMS due to carrier-imposed size restrictions.

Calendar	clicks	time (s)
Regular Calendar	10	49.7
Oracle Calendar	7	13.5
Copernicus	7	13.1

Email	clicks	time (s)
GMail (linear scan)	18	33.6
GMail (sender search)	14	27.5
GMail (subject search)	16	18.7
Copernicus	4	11.7

Photo	interactions		
	sender	receiver	time (s)
link (email)	15	6	66.2
link (SMS)	15	1	48.8
attachment (large,email)	10	6	96.8
attachment (small,email)	10	6	79.3
attachment (large,BT)	3	3	58.4
attachment (small,BT)	3	3	37.1
attachment (large,MMS) †	-	-	-
attachment (small,MMS)	14	3	48.8
Copernicus	3	4	15.0

1: *Results for Calendar, Email, and Photo. Large photo is 358k, small is 36k.* †*failed due to carrier's 250k limit.*

Copernicus significantly outperforms the alternative approaches requiring up to 65% fewer interactions and taking between 60% and 85% less time to complete the task.

6. RELATED WORKS

Previous research efforts have observed that mobile devices can provide potentially useful context based on absolute geographic location as well as other nearby devices or markers. This context can be tagged to content produced by mobile devices [15, 20]. This tagging can be used to provide other higher-level services, such as image recognition. Mobile Media Metadata [18] provided a framework in which photographs taken using the mobile phone are tagged with signatures detected by the device, and sent to a server which performs image recognition to further tag and group the photograph. The processed result is given back to the device for user acceptance and or correction. This work focuses on utilizing context information to facilitate effective sharing and collaboration over content and services accesible from mobile devices.

Device context has also been used to tailor the display of web content on mobile devices, which by their portable nature have constrained display surfaces and input modalities. This

has led to efforts in automatically adapting the content served by Internet services for mobile devices [3, 4, 13]. Many efforts have also attempted to improve the user interface on mobile devices, to better suit navigation, search, and selection of content returned from online services [1,11]. Chameleon [13], for example, utilizes viewer adaptation history in order to customize the presentation of images. CMo [4] utilizes machine learning on a proxy server to further digest and break down the content from a web page in order to serve it more succinctly and clearly. These works have mainly focused on manipulating the layout or focus of objects in the web browser or utilizing a specialized application to provide the user interface. Copernicus extends beyond these works by adapting web application functionality, not just content.

The use of context for adapting systems to mobile users is not a new concept. Copernicus utilizes Bluetooth device proximity because it is a tractable problem due to the pervasive availability of Bluetooth support on mobile devices. Copernicus's approach for adapting content has the potential for much greater impact now, due to the widespread use of large webapps and web services for hosting user content. Previous works have mostly been restricted to adapting content from disparate and disjoint machines and devices.

Tags found from other nearby mobile devices can be used to infer the social proximity of individuals correlated with the sensed devices and the location context [19]. Our work extends these efforts by utilizing the local proximity context of mobile devices to enhance the usage and interface experience of Internet web-based applications and services. Some results [16] have suggested that the assumed frequent closeness of users and their devices may not be as high when taking into account the whole day. We conjecture that for times when users are mobile to be in social settings, the closeness of devices to users will often hold true. Furthermore, our system does not completely depend on device proximity to achieve sharing and collaboration, since identity can also be specified using a 10 digit cellular number, even in the absence of one of the devices.

Many research efforts have explored the utility and use of social presence [6] for initiating the sharing of content, selecting the subset of applicable content, and determining the access rights and lifetimes of the share [10, 12, 17, 23]. The Cobalt [23] presented a system in which devices can automatically grant or revoke access rights based proximity. Push!Photo [17] presented a similar application of photo sharing based on social presence for content tagging and sharing. While this work shares a common motivating application with these other works, a significant distinction is that this work does not transfer content peer-to-peer between devices. The objective of this work is to enable proximity context filtering and searching for Internet based services in general. While the use of social presence [12] and collaborative filtering [9] has been previously suggested for improving services such as email, we believe this work contributes a novel study of an implementation for enabling context aware searching and filtering for Internet services.

7. CONCLUSIONS

In this paper we have presented Copernicus, a novel mobile system that enables sharing of web content and collaboration when users interact socially in the physical world. The key idea is that Copernicus allows adapting web applications so that they are aware of nearby users. We have built a prototype implementation of Copernicus. Our implementation supports a diverse range of applications hosted by different service providers, including photo and file sharing, calendar sharing, and email sorting, showing the applicability of our approach. Our evaluation shows that Copernicus reduces interaction times for finding, sharing, and accessing web content.

As future work, we envision using the Copernicus framework to build proximity-based applications that are currently not possible. For example, a calendar application can provide journal-like features, annotating the user's calendar with people the user met throughout their workday, or collecting business cards for new contacts. Similarly, email applications can provide filtering based on recently met users, or the information in their business cards, even if these applications are hosted by different providers.

For security, privacy, or legal reasons, various web applications may wish to run their own Copernicus-like service. A non-centralized design can mitigate the damage a user suffers if a particular service were compromised. We envision extending the use of delegation credentials to enable interoperability between Copernicus-like services. A cooperative API standard, similar to the OpenSocial effort [14], would enable accessing and searching data across trusted services.

8. REFERENCES

1. Y. Arase, T. Hara, T. Uemukai, and S. Nishio. Opa browser: a web browser for cellular phone users. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 71–80, New York, NY, USA, 2007. ACM.

2. P. Bahl and V. N. Padmanabhan. Radar: an in-building rf-based user location and tracking system. volume 2, pages 775–784 vol.2, 2000.

3. N. Bila, T. Ronda, I. Mohomed, K. N. Truong, and E. de Lara. Pagetailor: reusable end-user customization for the mobile web. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 16–29, New York, NY, USA, 2007. ACM.

4. Y. Borodin, J. Mahmud, and I. Ramakrishnan. Context browsing with mobiles - when less is more. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 3–15, New York, NY, USA, 2007. ACM.

5. M. Y. Chen, T. Sohn, D. Chmelev, D. Haehnel, J. Hightower, J. Hughes, A. LaMarca, F. Potter,

I. Smith, and A. Varshavsky. Practical metropolitan-scale positioning for gsm phones. In *Proceedings of the Eighth International Conference on Ubiquitous Computing (Ubicomp 2006)*, Lecture Notes in Computer Science, pages 225–242. Springer-Verlag, September 2006.

6. L. P. Cox, A. Dalton, and V. Marupadi. Smokescreen: flexible privacy controls for presence-sharing. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 233–245, New York, NY, USA, 2007. ACM.
7. http://wiki.developers.facebook.com/index.php/FQL.
8. http://www.flickr.com/services/api/flickr.photos.search.html.
9. D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.
10. A. Karlson, G. Smith, B. Meyers, G. Robertson, and M. Czerwinski. Courier: A collaborative phone-based file exchange system. Technical Report MSR-TR-2008-05, Microsoft Research, 2008.
11. A. K. Karlson, G. G. Robertson, D. C. Robbins, M. P. Czerwinski, and G. R. Smith. Fathumb: a facet-based interface for mobile search. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 711–720, New York, NY, USA, 2006. ACM.
12. J. Krumm and K. Hinckley. The nearme wireless proximity server. In *UbiComp 2004: Proceedings of the 6th International Conference on Ubiquitous Computing*, pages 283–300, 2004.
13. I. Mohomed, J. C. Cai, S. Chavoshi, and E. de Lara. Context-aware interactive content adaptation. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 42–55, New York, NY, USA, 2006. ACM.
14. http://code.google.com/apis/opensocial/.
15. S. N. Patel and G. D. Abowd. The contextcam: Automated point of capture video annotation. In *UbiComp 2004: Proceedings of the 6th International Conference on Ubiquitous Computing*, pages 301–318, 2004.
16. S. N. Patel, J. A. Kientz, G. R. Hayes, S. Bhat, and G. D. Abowd. Farther than you may think: An empirical investigation of the proximity of users to their mobile phones. In *UbiComp 2006: Proceedings of the 8th International Conference on Ubiquitous Computing*, pages 123–140, 2006.
17. M. Rost, M. Jacobsson, and L. E. Holmquist. Push!photo: Informal photo sharing in ad-hoc networks. In *UbiComp 2006: Proceedings of the 8th International Conference on Ubiquitous Computing*, 2006.
18. R. Sarvas, E. Herrarte, A. Wilhelm, and M. Davis. Metadata creation system for mobile images. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 36–48, New York, NY, USA, 2004. ACM.
19. T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. E. Smith, S. Consolvo, J. Hightower, W. G. Griswold, and E. de Lara. Mobility detection using everyday gsm traces. In *UbiComp 2006: Proceedings of the 8th International Conference on Ubiquitous Computing*, pages 212–224, 2006.
20. K. N. Truong and G. D. Abowd. Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In *Pervasive 2004: Proceedings of the 2nd International Conference on Pervasive Computing*, pages 140–157, 2004.
21. A. Varshavsky, E. de Lara, A. LaMarca, J. Hightower, and V. Otsason. Gsm indoor localization. *Pervasive and Mobile Computing Journal*, 3(6):698–720, 2007.
22. A. Varshavsky, A. Scannell, A. LaMarca, and E. de Lara. Amigo: Proximity-based authentication of mobile devices. In *UbiComp 2007: Proceedings of the 9th International Conference on Ubiquitous Computing*, pages 253–270, 2007.
23. K. Veeraraghavan, A. Myrick, and J. Flinn. Cobalt: separating content distribution from authorization in distributed file systems. In *FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies*, pages 29–29, Berkeley, CA, USA, 2007. USENIX Association.