Xshare: Supporting Impromptu Sharing of Mobile Phones

Presented by: Wanlu Wang & Yi Liu

UCL Computer Science

Mobile and Cloud Computing
Presentation Outline

Part 1 (Wanlu Wang):
• Motivation & Related Work -- why
• Contribution -- what
• Design -- how

Part 2 (Yi Liu):
• Implementation -- by what
• Evaluation -- explain results
• Conclusion -- future work
Motivation

• It is often desirable to share phones with others:
  1. Social reason: Feature-rich & content (e.g. pictures) with others
  2. Community reason: price issue *(not anymore?)*

• Access control is **nothing or everything**
  – Set password for the entire system only
Related Work

• Technology Sharing: *face-to-face media sharing* (T.Pering et al) -> **no solution**

• Virtual Machine: isolating multiple VMs & preventing data altering -> **xShare based on single VM from accessing non-shared data/app**

• Support for Multiple Users: conventionally designed for different known users -> **xShare single user**

• Windows CE Kiosk Mode: Specific hardware dependent support mode -> **very limited**
Contribution

• Motivational user study: International Interview
  1. Phone sharing statistics: 93% lender/borrower
  2. Frequency of phone sharing: over 50% monthly
  3. Applications involved in phone sharing: must be scalable for a wide range of applications
  4. Relationships involved in phone sharing: customizable access to functionalities and data
  5. Locations of phone sharing: data may be confidential or sharable
  6. Reasons for sharing: data/application modification
  7. Reasons against sharing: over 90% privacy concerns (messages, photo/videos)
Contribution

• Motivational user study: Long-Term Field Study
  – All participants actively shared the mobiles as a social networking tool
  – Sharing decreased as the study went on (more privacy sensitive data)
  – Sharing is mostly impromptu, place/time of convenience, without prior planning
  – Application-driven and data-driven
  – Sharing policy is dynamically determined
Design Goal & Requirements

- **Goal**: a privacy solution must support impromptu configuration for a temporary guest user with minimum latency.
  - *not about security flaws but limitation on shared services and data*

- **Requirements**:
  - Impromptu policy creation
  - Access control
  - Resource accounting
  - Borrower data reconciliation
Design Overview

- File based access control
  - Files as an abstraction for data and app
  - Independent files -> independent app
  - no need to rebuild ROM image
User Interface

• Policy Specification: files, applications and resource allowance.
  – Select shared files and automatically select appropriate applications. *(can be overridden when selecting shared applications)*
  – All items default not shared
  – Read, write and create new files access granted
  – Profiles for frequently used sharing policies
  – Quick Share: only access to current front application and files opened
User Interface

• Shell Customization: shell UI shows some data (e.g. calendar events) or links to launch applications.
  – Simply hide the data/icons/links

• Borrower Data Reconciliation: manage changes made to files.
Access Control

• In-Memory Services & Applications:
  – OS keeps some services/applications in memory for faster launch -> **private data loaded already**
  – Simply terminates corresponding processes before entering Shared mode & re-launch them later (**Managing Unstoppable Services**)

• Identifying Files for Application Sharing:
  – Applications needs more than their executable files to run (e.g. config, DLL) -> **distinguish necessary files or private data**
  – Different folders of app files and data, set all the files in the same folder executable

• Implementing Access Control to Files:
  – Container file -> loaded into memory & binary search
Virtual Environment

• Namespace Virtualization:
  – Control resources access by renaming them (concealing files, redirecting write access)

• Change Separation:
  – Create a private folder to hold all the modified and created files instead of operating on original files (copy-on-write)
Resource Accounting

• As mentioned in design requirements, issues about exhaustible system resources (e.g. battery, storage, chargeable features)

• Using OS provided APIs to leverage these problems
  – Enable the owner to set restrictions: 5 MB storage, stopping sharing when battery below 20%
Data access control

- Challenge: No file-level access control in Windows Mobile OS.
  - xShare uses API interception.

- Application uses system API to grant the access to the data.
  - Systems APIs in server processes
  - Two kinds of system APIs WM-OS:
    - Implicit System APIs.
    - Handle-Based System APIs.
Implicit System APIs

- Implicit system API is globally registered in system API table, e.g. CreateFile()
Handle-Based System APIs

- Handle-Based API is associated with kernel object like file handle, e.g. `WriteFile()`
API Interception

Figure from xShare paper MobiSys’ 09 June 22 -25
Virtualization Environment

- Separate changes made in shared mode.
  - Physical path: path in normal mode
  - Virtual path: path used by apps
  - Intermediate path: prefix with “\xShare\Root”.
  - xShare saves the change in the intermediate path (copy on write).
  - xShare also creates a virtual link file at each intermediate path for the mapping.
Name Virtualization

- **Hiding non-shared file**
  - xShare uses API interception routine again to hide the files.
  - It intercepts `FindFirstFile()` and `FindNextFile()` to make unshared file invisible in shared mode.
  - **Consistent view:** Search through both physical path and intermediate path.
Evaluation

• Testing xShare on HTC Wizard phone
• Evaluate xShare in two aspects
  - Measurement.
    - Latency, execution overhead and energy.
  - User studies.
    - Owner.
    - Borrower.
Latency Measurement

- Launch a branch application in the background.
  50 SMSes, Contacts, Messages, Solitaire, ActiveSync and Notes.
- Repeat the experiment 10 times.
- It costs 5.8 seconds on average to switch to share mode.
- It costs 3 seconds to switch back to normal mode.
- Latency is fine (from user studies).
- xShare has litter impact on the overall performance and energy.
- But it still has a significant overhead when open file for write and create new file in shared mode as xShare uses copy on write.
Future Work

• **Improve security**
  - xShare is still breakable.
  - exploit system flaws.
  - hack into the system.

• **Implement xShare on other OS**
  - xShare highly relies on OS
  - may have issues when implement xShare on other OS

• **Limit Latency**
  - If owner runs a lot of big apps on the background. *e.g* 3D game.
  - If owner has large number of emails or SMSes in the smart phone.
Conclusion

• Phone sharing is popular in social community
  - Wide range of apps
  - People concern privacy in the smart phone

• xShare:
  - Sharing policy
  - Data Access Control
  - Virtual Environment

• **Pro:** User friendly, lightweight and Security.

• **Con:** Highly rely on OS, Latency.