FLUID: Flexible User Interface Distribution for Ubiquitous Multi-device Interaction

Sangeun Oh*, Ahyeon Kim*, Sunjae Lee*, Kilho Lee*, Dae R. Jeong*, Steven Y. Ko†, and Insik Shin*

*KAIST
†University at Buffalo, The State University of New York
Various surfaces become pervasive!

• Smart devices have various surfaces with different shapes & sizes
  – From smartwatch to smart TV
  – Foldable screen (Samsung Galaxy Fold) / dual screen (LG V50)
Potential for multi-surface interaction

• The trend can change how users interact with applications
  – Using only single surface ➔ Using multiple surfaces concurrently
Potential for multi-surface interaction

• The trend can change how users interact with applications
  – Using only single surface ➔ Using multiple surfaces concurrently
Use case: live streaming

Twitch  LiveMe
Instagram  VK live
Existing solutions

- **Customized apps**
  - Extra engineering efforts
  - Low applicability

- **Screen mirroring**
  - Low flexibility
    - Supports only full screen
  - Low responsiveness for high resolutions

- **App migration**
  - Low flexibility
    - Supports only full screen
    - Cannot support concurrent usage

Flux [EuroSys'15]
Research goal

• Design a new mobile platform that supports multi-surface interaction by distributing UI objects to different devices
  – in a flexible, transparent and responsive manner

FLUID
(Flexible UI Distribution)
FLUID overview

- **Key idea**: separation between app logic & UI parts
**FLUID overview**

- **Key idea**: separation between app logic & UI parts
  1) Distributing target UI objects to remote devices and rendering them

- Diagram:
  - **Target App**
    - App logic
    - UI object A
    - UI object B
    - Surface
    - <Host device>
  - **FLUID Wrapper App**
    - Surface
    - <Guest device>
  - UI selection

- "UI selection" arrow connecting UI object A from the Target App to the FLUID Wrapper App.
**FLUID overview**

- **Key idea**: separation between app logic & UI parts
  1) Distributing target UI objects to remote devices and rendering them

![Diagram of FLUID overview]

- Target App
  - **App logic**
  - **UI object A**
  - **UI object B**
  - **Surface**
  - **<Host device>**

- FLUID Wrapper App
  - **UI object A**
  - **Surface**
  - **<Guest device>**

UI distribution
**FLUID overview**

- **Key idea**: separation between app logic & UI parts
  1) Distributing target UI objects to remote devices and rendering them
  2) Giving an illusion as if app logic and UI objects were in the same process
Why is FLUID good?

• **Flexibility**
  – Allow users to control multiple surfaces as they want via fine-grained UI distribution

• **Transparency**
  – Support legacy apps without any modification to them
  – Develop new multi-surface apps under the existing programming model

• **Responsiveness**
  – Require less network transmission when moving UIs instead of full screen
Problems

• **P1.** How to split & distribute UI objects?
  - Transmits minimum graphical states needed for UI rendering
    - To reduce network overhead
  - However, it is unknown which graphical states app-specific custom UIs use
Problems

• **P1.** How to split & distribute UI objects?
  - Transmits minimum graphical states needed for UI rendering
    ▪ To reduce network overhead
  - However, it is unknown which graphical states app-specific custom UIs use
  - **Our solution:** Selective UI distribution
• **P2.** How to maintain interaction between app logic & UI objects?
  
  – Such interaction is achieved via local function calls
    
    ▪ e.g., `TextView.setText()`, `ImageView.setImageResource()`, etc.
  
  – However, local functions cannot be executed across devices
  
  – **Our solution:** transparent RPC transformation in Android VM (ART)

![Diagram](image-url)
Evaluation environment

• Implemented FLUID prototype based on Android 8.1 (Oreo)
• Used Google Pixel XL (smartphone) & Pixel C (tablet)
  – Phone-to-phone
  – Phone-to-tablet
  – Tablet-to-phone
• On the Same WiFi network
App coverage

• Using 20 legacy apps for 10 multi-surface scenarios
  – All legacy apps use their own custom UIs
• FLUID can support various legacy apps successfully

<table>
<thead>
<tr>
<th>Use case scenario</th>
<th>UI type</th>
<th>App name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login with personal device</td>
<td>Editor</td>
<td>Instagram, Paypal</td>
</tr>
<tr>
<td>Fill in information collaboratively</td>
<td>Text, editor</td>
<td>eBay, Booking.com</td>
</tr>
<tr>
<td>Chatting with different device while broadcasting</td>
<td>Button, editor</td>
<td>LiveMe, Afreeca TV</td>
</tr>
<tr>
<td>Search destination with different device</td>
<td>Button, editor</td>
<td>Naver map, Maps.me</td>
</tr>
<tr>
<td>Control media with different device</td>
<td>Seek bar, button</td>
<td>VLC Player, Music Player</td>
</tr>
<tr>
<td>Control painting tool with different device</td>
<td>Scroll, image, button</td>
<td>PaperDraw, Paint</td>
</tr>
<tr>
<td>Sharing photo to public device</td>
<td>Image</td>
<td>Gallery, A+ Gallery</td>
</tr>
<tr>
<td>Unlock pattern with personal device</td>
<td>Pattern lock</td>
<td>Smart app lock, AppLock</td>
</tr>
<tr>
<td>Read document with different device</td>
<td>Text, scroll</td>
<td>File Viewer, Bible KJV</td>
</tr>
<tr>
<td>Edit text on different device</td>
<td>Editor</td>
<td>Color note, Text editor</td>
</tr>
</tbody>
</table>
UI distribution time

- It ranges from 132 to 735ms ➔ Fast enough for interactive use
Network transfer over time

- Comparing transfer pattern of FLUID and other mirroring techniques
  - Under the same scenario that a user types destination (11 characters) into the search box UI of Naver map
Conclusion

• Designed & implemented FLUID
  – Separation between app logic & UIs
  – Evaluated with 20 legacy apps for 10 multi-surface scenarios

• Expect FLUID to accelerate development of creative and useful apps to provide novel UX
Thank you!

Visit cps.kaist.ac.kr/fluid for more information:)