



30-Day Android Challenge

From Basics to Advanced

(Expert-Level Deconstruction 📱)

Day 6

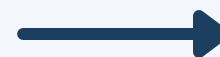
Intents & Intent Filters

Android Intent Architecture: From Component Communication to AOSP Lower Layers 🚀



Esmaeeil Moradi

@esmaeeil-moradi700



Day 6

Demystifying Android Intents: From Component Communication to AOSP Depths

Welcome to the community of Android architects and seasoned enthusiasts!

Hey Android aficionados!

Today, on the sixth day of our challenge, we delve into the heart of Android's communication system: Intents and Intent Filters. This concept is not only fundamental to interaction between components but also exemplifies messaging-based design in AOSP. In this post, we will examine Intents from the perspective of operating system architecture, performance, security, and scalability.



Esmaeel Moradi

@esmaeeil-moradi700



Intents: Smart Communication Packets

Intents in Android function like smart data packets containing execution instructions for the system. Each Intent comprises the following information:

- **Action:** What's the mission? (e.g., VIEW, SEND)
- **Data:** What resources are involved? (URIs, MIME types)
- **Category:** How does this fit into the bigger picture? (e.g., DEFAULT, BROWSABLE)
- **Component:** Who's the direct recipient? (Explicit Intents only)
- **Extras:** Any bonus info to include? (Key-value pairs)
- **Flags:** How should this be handled? (e.g., back stack behavior)



Esmaeel Moradi
@esmaeel-moradi700



Intents in Action: Explicit vs. Implicit

Explicit Intents: Direct Delivery

- Target: Precisely defined – the exact component to activate (package + class name).
- Use Case: Tight coordination within your app, or launching a specific activity in another app.
- How it Works: Android shortcuts the Intent Filter process and delivers straight to the target.
- Code Example (Kotlin):

// Example: Launching an activity within the app

```
val intent = Intent(this, DetailActivity::class.java)
startActivity(intent)
```

- **AOSP Insight (ActivityStarter.java):**

```
if (intent.getComponent() != null) {
    mSupervisor.startActivitySpecific(...);
}
```



Esmaeel Moradi

@esmaeeil-moradi700



Implicit Intents: Finding the Right Handler

- Target: Flexible – relies on the system to find the best-suited app or component.
- Use Case: Triggering actions without needing to know the exact app (e.g., sharing content, opening web pages).
- How it Works: Android's IntentResolver steps in to match the Intent's details against registered Intent Filters.

```
// Example: Sharing an image
val intent = Intent(Intent.ACTION_SEND).apply {
    type = "image/*"
    putExtra(Intent.EXTRA_STREAM, imageUri)
}
startActivity(intent)
```

- **AOSP Insight (IntentResolver.java):**

```
List<ResolveInfo> resolveInfos =
    queryIntentActivities(intent, flags);
```



Esmaeel Moradi
@esmaeeil-moradi700



Intent Filters: Defining What Your Components Accept

- Intent Filters, declared in your app's manifest, are like the gatekeepers for activities, services, and broadcast receivers. They specify the kinds of Intents a component is willing to handle.

- XML Example:

```
<activity android:name=".ShareActivity"> <intent-filter>  
<action android:name="android.intent.action.SEND" />  
<category  
android:name="android.intent.category.DEFAULT" /> <data  
android:mimeType="image/*" /> </intent-filter> </activity>
```

This filter enables ShareActivity to handle image sharing requests from any app!

This means ShareActivity can receive image sharing requests from any application.



Esmaeel Moradi

@esmaeel-moradi700

Explicit vs. Implicit Intents Comparison

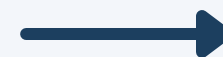
Table

Feature	Explicit Intent	Implicit Intent
Target	Specific (Package + Class)	Unspecified (based on Action)
Security	High (no permissions needed)	Medium (permissions may be needed)
AOSP Usage	Direct (Binder Call)	IntentResolver
Resource Consumption	Low	Moderate (pattern matching)
Case Study	Launching internal Activity	Sharing between applications



Esmaeel Moradi
@esmaeel-moradi700

Intent Resolution Architecture in AOSP



The process of matching Intents in AOSP involves the following steps:

- Candidate Collection:

Using PackageManagerService to retrieve the list of registered components

- Initial Filtering:

Removing components without necessary permissions or incompatible with the platform

- Action and Category Matching:

IntentResolver first matches the Action and then the Category.

- Data Matching (MIME type or URI):

If data is present, it must match the data defined in the Filter.

- Scoring and Prioritization:

An algorithm based on the following priorities:

- Exact match of Action and Category
- Match of Data Type and URI
- Priority specified in the manifest (priority)

- Final Selection:

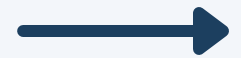
If there is only one option, it is executed; if there are multiple options with equal scores, the system prompts the user to choose.



Esmaeel Moradi

@esmaeel-moradi700

Simplified code from IntentResolver in AOSP:



```
List<ResolveInfo> resolveInfos = queryIntentActivities(intent, flags);  
// Filtering and sorting based on score and priority
```

Intent as a Binder Token: Cross-Process Mechanics (AOSP)

- When an Intent crosses process boundaries (e.g., from an application to system_server), it is converted into a Parcel object. The Binder driver serializes/deserializes the Intent using the writeToParcel() and CREATOR methods.

```
public void writeToParcel(Parcel out, int flags) {  
    out.writeString(mAction);  
    out.writeIdentifier(mPackage); // Critical for UID checks  
    out.writeUri(mData);  
    // ... Marshaling of extras, flags, etc.  
}
```



Esmaeel Moradi
@esmaeeil-moradi700

ActivityManagerService (AMS) as the Intent Router (AOSP)

- **AMS (ActivityManagerService.java) acts as the central dispatcher for all Activity-related Intents.**
- **Critical Path:**
 - startActivity() → ActivityTaskManagerService (ATMS)
 - ATMS resolves the target via resolveIntent() → PackageManagerService (PMS)
 - PMS queries IntentResolver for matches
 - AMS enforces permissions via checkStartActivityResult()
- **Security Check (AOSP):**

```
// ActivityManagerService.java
final int startResult = startActivityAsUser(...);
if (startResult != ActivityManager.START_SUCCESS) {
    throw new SecurityException("Permission denied");
}
```



Esmaeel Moradi
@esmaeel-moradi700



Intent Resolution: AOSP's IntentResolver Deep Dive (AOSP)

- For scalability, PMS maintains in-memory caches (mActivities, mServices) of all registered IntentFilters, indexed by:
 - Action strings (hash map)
 - MIME types (partial trie)
 - Categories (hash map)
- Indexing Logic (IntentResolver.java):

```
// Pre-filtering by action
List<F> quick = mActionToFilter.get(action);
if (quick != null) {
    // Secondary filtering by data/category
}
```



Esmaeel Moradi

@esmaeeil-moradi700



Algorithm Complexity:

- Best Case: $O(1)$ for explicit intents or cached resolutions.
- Worst Case: $O(N)$ for implicit intents with broad filters (e.g., ACTION_VIEW with http scheme).
- Bottleneck: Linear scans for wildcard MIME types (image/*) or URI path patterns.

Optimization Strategies (OEM-Level):

- Bloom Filters for Action/Category: Reducing false positives during initial filtering.
- Trie for MIME Types: Hierarchical matching for type/subtype patterns.
- Pre-resolved Intent Caches: Caching frequently used implicit intents (e.g., browser, dialer) in Zygote.



Esmaeel Moradi

@esmaeel-moradi700

Practical and Security Scenarios



- Sending Email with Implicit Intent:

```
val intent = Intent(Intent.ACTION_SENDTO)
intent.data = Uri.parse("mailto:example@example.com")
startActivity(intent)
```

- Sharing an Image:

```
val intent = Intent(Intent.ACTION_SEND)
intent.type = "image/*"
intent.putExtra(Intent.EXTRA_STREAM, imageUri)
startActivity(Intent(Intent.ACTION_SEND, intent, "Share Image"))
```

- Checking for a Receiver:

```
if (intent.resolveActivity(packageManager) != null) {
    startActivity(intent)
}
```



Esmaeel Moradi
@esmaeel-moradi700

Important Security Notes:



- Do not send sensitive data in Implicit Intents unless you are sure the receiver is trustworthy.
- Use Explicit Intents for critical operations or verify the signature of the target application.
- Always use `intent.resolveActivity()` to check for the existence of a receiver.
- For sensitive operations, use `PackageManager.queryIntentActivities()` and validate the signature.

Security Hardening: Attack Vectors and Mitigations (AOSP)

- **Intent Spoofing & Injection:**
- Vulnerability: A malicious application declares a high-priority filter for common actions (e.g., `ACTION_SEND`).
- Mitigation:
- Use `Intent.setPackage()` to restrict resolution to trusted applications.
- Validate the signature of the resolved component using `PackageManager.getPackageInfo()`:

```
val info = packageManager.getPackageInfo(targetPackage,  
PackageManager.GET_SIGNATURES) if (info.signatures[0] !=  
trustedSignature) { /* Block */ }
```



Esmaeel Moradi

@esmaeel-moradi700

PendingIntent Hijacking:



- Risk: An old PendingIntent is reused by malicious applications.
 - Mitigation:
 - Always use FLAG_IMMUTABLE unless mutability is required.
 - Attach a WhitelistToken (internal to AOSP) to restrict receivers.
-
- AOSP Code (PendingIntent.java):

```
// FLAG_IMMUTABLE ensures that the PendingIntent cannot be modified
public static PendingIntent getActivity(..., int flags) {
    if ((flags & FLAG_MUTABLE) != 0) {
        enforceMutabilityRestrictions(); // Throws SecurityException
    }
}
```



Esmaeel Moradi
@esmaeeil-moradi700

Intents and Broadcasts



In addition to launching Activities or Services, Intents are also used to send and receive Broadcasts (e.g., receiving SMS or network state changes).

Custom Intent Actions (AOSP)

System applications can declare privileged intent filters in /system/priv-app manifests. These bypass normal permission checks.

- Example (CustomOemService):

```
<intent-filter android:priority="1000">  
  <action android:name="com.oem.CUSTOM_ACTION" />  
  <category android:name="android.intent.category.DEFAULT" />  
</intent-filter>
```



Esmaeel Moradi

@esmaeeil-moradi700



Dynamic Intent Filter Registration (AOSP)

While most filters are static, runtime registration is possible via `PackageManager.addFilter()` (requires `SIGNATURE` permission):

- System-level code only:

```
IntentFilter filter = new IntentFilter("com.internal.ACTION");  
mPackageManager.addFilter(filter, "com.internal.PERMISSION");
```



Esmameil Moradi

@esmaeeil-moradi700

Explicit vs. Implicit Intents Comparison

Table2

Feature	Explicit Intent	Implicit Intent
Target	Specific (Component Name)	Unspecified (Based on Action)
Intent Filter Required	No	Yes
Security	High	Medium (Filter-dependent)
Usage	Internal, Direct	General, Inter-application
Performance	Fast	Slower (Needs Resolution)



Esmaeel Moradi
@esmaeel-moradi700

Performance Metrics and Debugging (AOSP) —→

Systrace & ATMS Logs:

- Systrace Tag: am_resolve_intent indicates time spent in resolution.
- Debugging

```
adb shell dumpsys activity intents # Dump all registered filters
adb shell dumpsys package queries # Inspect PMS query history
```

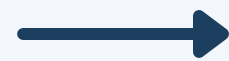
Benchmarking Intent Resolution:

```
val start = SystemClock.uptimeMillis()
val resolved = intent.resolveActivity(packageManager)
val duration = SystemClock.uptimeMillis() - start
Log.d("Perf", "Intent resolution took $duration ms")
```



Esmaeel Moradi
@esmaeeil-moradi700

? Advanced Interview Insights



- **What is the difference between PackageManager and IntentResolver in AOSP?**

PackageManager is responsible for managing package information, while IntentResolver executes the Intent matching algorithm.

- **How does the system use Memory Mapping for Intents?**

Intent data is transferred Memory-Mapped in Binder Transactions.

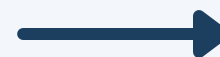
- **Can Intent Filters be registered dynamically?**

Yes, using PackageManager.addFilter() but requires system-level SIGNATURE permission.



Esmaeel Moradi

@esmaeel-moradi700



Challenge Question for AOSP Experts

Given that the Intent Resolution process in AOSP uses a linear search algorithm, how can this process be optimized for applications with hundreds of Intent Filters? Is it feasible to use advanced data structures like a Trie Tree?



Esmaeel Moradi

@esmaeeil-moradi700

Thought Experiment: Redesigning Intent Resolution (AOSP)

- Problem: $O(N)$ linear scans in PMS become prohibitive on devices with 500+ apps.
- Proposed Solution:
 - Hierarchical Intent Tree:
 - Root: Action $\rightarrow$ Category $\rightarrow$ MIME Type $\rightarrow$ URI
 - Each node uses hash maps or tries for $O(1)$ lookups.
 - Precomputed Wildcard Matches:
 - Cache image/* matches to avoid redundant scans.
 - Trade-offs:
 - Memory Overhead: Indexes might consume 10-15% more RAM.
 - Zygote Preloading: Requires AOSP modifications to precompute trees.



Esmaeel Moradi

@esmaeel-moradi700



Conclusion and Call to Action

- Intents play a crucial role in Android architecture. They not only enable communication between different components of an application but also allow applications to interact with other programs. A deep understanding of the Intent and Intent Filter mechanism helps developers create more flexible, efficient, and secure applications.
- We invite you to share your thoughts, experiences, and suggestions on this topic. Let's elevate our knowledge in Android development together!



Esmaeel Moradi
@esmaeeil-moradi700

Looking forward to your
insights and experiences!
Let's elevate Android
development together.



LEAVE A COMMENT BELOW



Esmaeeil Moradi

@esmaeeil-moradi700