

# Exercice Scientifique et Technique

## AIDL : Communication Inter-Processus dans Android

### LANNUZEL Dylan

« This document and the information contained here in are the property of Safran. They must not be copied or disclosed to third parties without prior Safran written authorization;

Ce document et les informations qu'il contient sont la propriété de Safran. Ils ne doivent être copiés ni communiqués à un tiers sans autorisation préalable et écrite de Safran. »



# Sommaire

---

**1. Introduction à Android**

**2. Problématique : l'IPC sur Android**

**3. Présentation de l'AIDL**

**4. Fonctionnement détaillé**

**5. Exemple de code AIDL**

**6. Alternatives : Messenger, Broadcast, ContentProvider**

**7. Tableau comparatif**

**8. Cas d'usage et bonnes pratiques**

**9. Conclusion et références**

« This document and the information contained here in are the property of Safran. They must not be copied or disclosed to third parties without prior Safran written authorization;

Ce document et les informations qu'il contient sont la propriété de Safran. Ils ne doivent être copiés ni communiqués à un tiers sans autorisation préalable et écrite de Safran. »

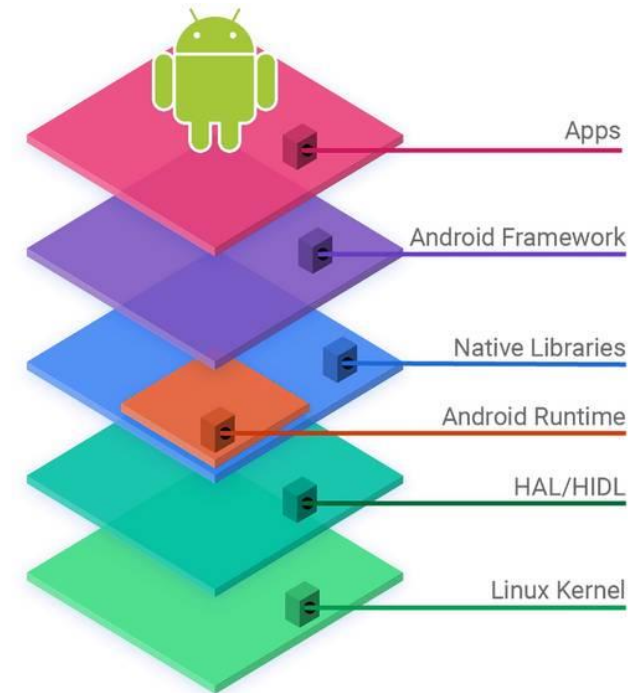
# 1. Introduction à Android

## Qu'est-ce qu'Android ?

- Système d'exploitation mobile open source (Google)
- Basé sur le noyau Linux
- ~72% de parts de marché mondial
- **Architecture en couches**
- Chaque application tourne dans son propre processus

## Isolation des processus

- Sécurité : sandbox par application
- Nécessité de mécanismes d'IPC



## 2. Problématique : la communication inter-processus (IPC)

### Le problème

- Un processus ne peut pas accéder directement à la mémoire d'un autre processus
- Les objets doivent être décomposés en primitives pour traverser les frontières de processus
- La sérialisation est complexe et source d'erreurs

### Enjeux

- Sécurité, performance, fiabilité

### Cas concrets nécessitant l'IPC

- Application de musique : UI + service de lecture en arrière-plan
- SDK de paiement exposant des API à d'autres apps
- Services système (téléphonie, capteurs, etc.)

### Le framework Binder

- Mécanisme IPC natif d'Android (driver noyau)
- Tous les mécanismes IPC Android (Broadcast, Messenger, AIDL) reposent sur Binder

# 3. Présentation de l'AIDL

## Définition

- AIDL = Android Interface Definition Language
- Langage de définition d'interface pour l'IPC sur Android
- Plus léger et adapté à l'écosystème que HIDL ou CORBA

## Principe

- Définir un contrat (interface) entre client et service
- Génération automatique du code de sérialisation (Stub/Proxy)

## Quand utiliser l'AIDL ?

- IPC entre applications différentes
- Besoin de requêtes concurrentes
- Transfert de types complexes (Parcelable)
- Besoin de maintenir le service Android éveillé

## Types de données supportés

- Primitifs Java (int, long, boolean, char...)
- String, CharSequence, List, Map
- Objets Parcelable personnalisés

## 4. Fonctionnement détaillé de l'AIDL

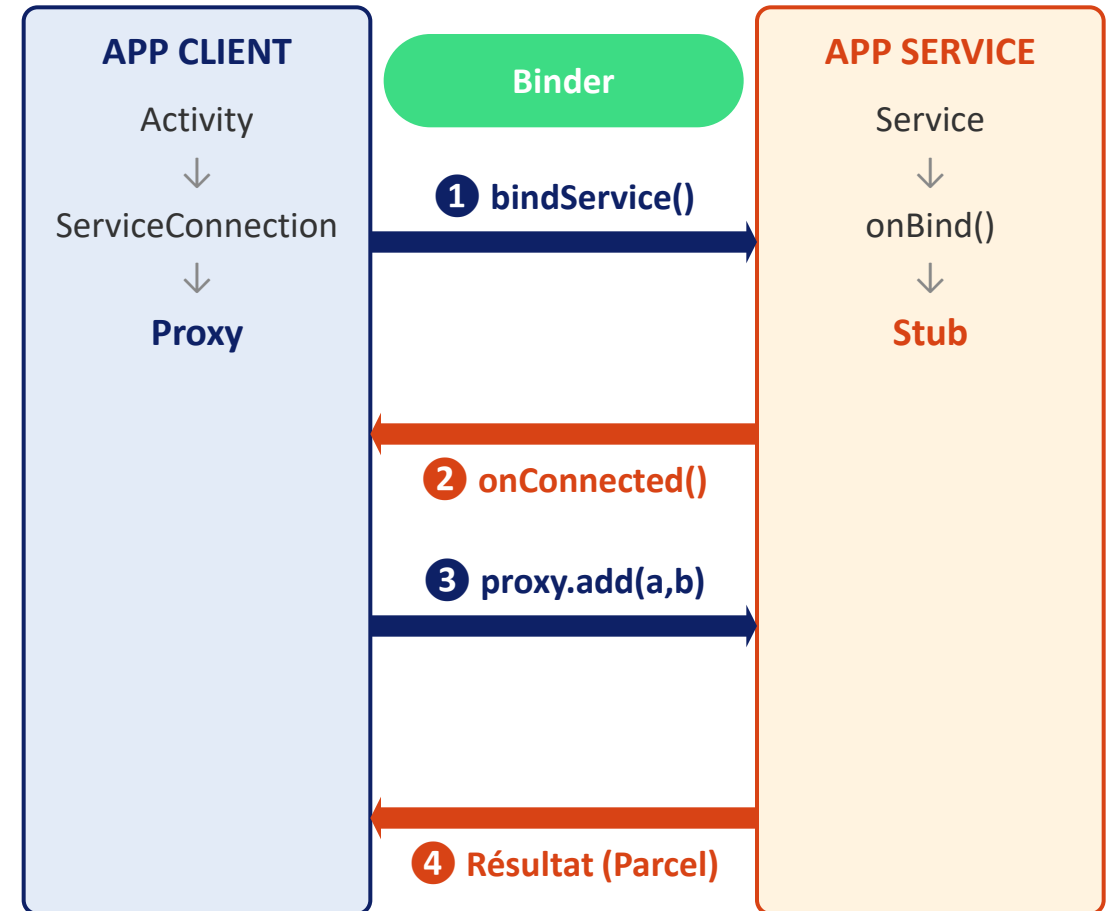
### Étapes de mise en œuvre

1. Créer le fichier .aidl
2. Le SDK génère les classes Stub et Proxy
3. Implémenter l'interface dans le Service
4. Le client se lie via `bindService()`, le service se réveille et reste éveillé
5. Appeler les méthodes via `Ibinder`

### Balise directionnelle pour les types non primitifs

- **in** (client → service)
- **out** (service → client)
- **inout** (bidirectionnel)

### Flux d'un appel AIDL



## 5a. Définition AIDL + Service

```
// CalcRequest.aidl
parcelable CalcRequest {
    int a;
    int b;
}

// ICalculator.aidl
import com.example.CalcRequest;

interface ICalculator {
    int add(int a, int b);

    oneway void compute(
        in CalcRequest req);
}
```

```
// CalcService.kt (serveur)
class CalcService : Service() {
    val binder = object :
        ICalculator.Stub() {

        override fun add(
            a:Int, b:Int) = a + b

        override fun compute(req: CalcRequest) {
            Log.d("Calc",
                "${req.a}+${req.b}")
        }
    }
    override fun onBind(i: Intent) = binder
}
```

## 5b. Exemple : Client (binding + appel)

```
// Client
var calc: ICalculator? = null
val conn = object :
    ServiceConnection {
        override fun onServiceConnected(
            n: ComponentName,
            svc: IBinder) {
            calc = ICalculator.Stub.asInterface(svc)
        }
    }
// Binding + appels
bindService(intent, conn, BIND_AUTO_CREATE)
val r = calc?.add(3, 5)
calc?.compute(CalcRequest(3, 5))
```

### Explication du code

- ServiceConnection reçoit le callback onServiceConnected
- Stub.asInterface() retourne le Proxy
- add() : appel synchrone avec retour
- compute() : appel oneway (asynchrone)

### Points clés

- CalcRequest est un Parcelable (tag in)
- oneway : le client n'attend pas de réponse
- Thread-safety à garantir côté implémentation des interfaces



## 6. Alternatives à l'AIDL

### Messenger

- Plus simple qu'AIDL, basé sur Handler/Message
- Traitement séquentiel (mono-thread)
- Pas de type safety, adapté aux petites données

### Broadcast (BroadcastReceiver)

- Communication asynchrone de type publish-subscribe
- Idéal pour les événements système (batterie, réseau)
- Pas adapté aux données volumineuses

### ContentProvider

- API CRUD (Create, Read, Update, Delete)
- Adapté au partage de données structurées (BDD, fichiers)
- Communication synchrone via URIs

### IPC réseau

- Websockets, API Web, gRPC
- Extérieur à l'écosystème Android, le service peut être tué par le système
- Gestion des accès et authentification

## 7. Tableau comparatif des mécanismes IPC

Critère	AIDL	Messenger	Broadcast
Req. parallèles	Oui	Non	Non
Type safety	Oui (typé)	Non (Message)	Non (Bundle)
Complexité	Moyenne	Faible	Faible
Cas d'usage	IPC multi-client, API structurée	IPC simple, mono-thread	Événements système

## 8. Cas d'usage et bonnes pratiques

### Cas d'usage typiques

- Applications de lecture média (UI + service de lecture dans des processus séparés)
- SDK tiers exposant des services (paiement, authentification)
- Services système Android (téléphonie, capteurs)
- Apps multi-processus avec appels concurrents

### Bonnes pratiques

- Garder les interfaces stables (éviter les breaking changes)
- Assurer la thread-safety (synchronized, locks)

***Utiliser l'AIDL si une API structurée est requise***

## 9. Conclusion et Références

### Synthèse

- L'IPC est un enjeu majeur de l'architecture Android
- AIDL est la solution la plus puissante pour l'IPC typé avec requêtes parallèles
- Mais Messenger ou Broadcast peuvent suffire
- Le framework Binder reste le socle commun de tous ces mécanismes

### Lien avec l'enseignement

- Programmation orientée objet
- Android

### Références bibliographiques

- [1] Android Developers – AIDL : [developer.android.com/develop/background-work/services/aidl](https://developer.android.com/develop/background-work/services/aidl)
- [2] AOSP – AIDL Overview : [source.android.com/docs/core/architecture/aidl](https://source.android.com/docs/core/architecture/aidl)
- [3] S. Bazhenov – Mastering Inter-Process Communication (IPC) in Android : [proandroiddev.com/mastering-inter-process-communication-ipc-in-android-fe471c7796e4](https://proandroiddev.com/mastering-inter-process-communication-ipc-in-android-fe471c7796e4)
- [4] Android Developers – Bound Services : [developer.android.com/develop/background-work/services/bound-services](https://developer.android.com/develop/background-work/services/bound-services)
- [5] How to Use Android Without Google: Everything You Need to Know : [makeuseof.com/tag/using-android-without-google/](https://makeuseof.com/tag/using-android-without-google/)

***Merci pour votre attention !***