안드로이드 바인더

안드로이드 RPC 메커니즘의 이해

오 동권
www.flowdas.com

2009 제4회 Korea Android 세미나

FLOWDAS

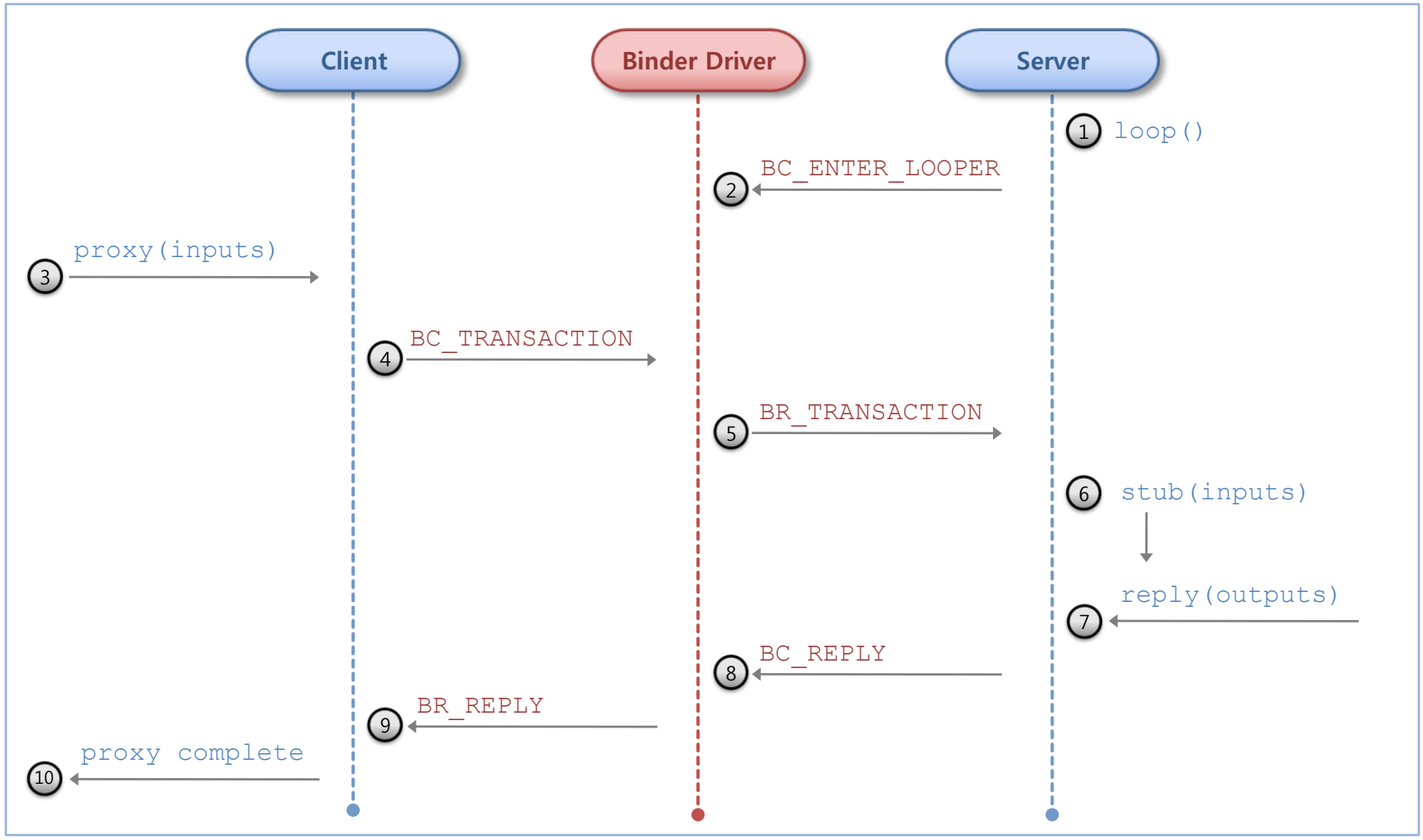
The Void Which Binds*

* "Hyperion" by Dan Simmons

OpenBinder

- Dannie Hackborn
- OpenBinder
  - 차세대 BeOS를 위한 내부 프로젝트로 시작
    - Hardware Scalability
    - System Customization
    - Robust Applications
  - PalmOS 6 Cobalt에서 처음으로 구현
  - Linux 구현이 오픈 소스로 공개 (C++)
  - 2005년 개발 중단
- Android Binder
  - Android의 표준 IPC
  - Google의 재구현 (대부분 Java)
  - Kernel Module은 OpenBinder 코드 재사용
  - OpenBinder와 설계 철학을 공유

Remote Procedure Call



Thread Model

- Client가 LPC와 RPC의 차이를 구분하지 못하도록 하는 것이 목표.
- Thread Migration Model
  - Call
    - Caller의 Thread에서 실행되던 코드는 Callee의 Thread 로 이동하여 실행
    - Callee가 User Space에서 Thread Pool을 운용.
    - Thread Pool의 크기 제한은 없으나 부족하면 ANR 가능.
  - Return
    - 원래의 Caller Thread로 복귀하도록 Binder Driver가 지원.
  - Transparent Recursion
    - Callee가 Caller의 Callback을 호출하면, Call을 시작한 Caller의 원래 Thread에서 실행.
    - Binder Driver가 Call History를 추적.

/dev/binder

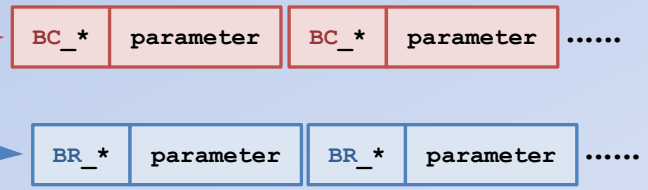
- Binder에 특화된 IPC Mechanism을 제공.
  - Misc Device Driver
- `<linux/binder.h>`
  - NDK에 포함되어 있으나 Unstable.
  - Protocol Version 7
- `ioctl` commands

Request	Data type	R/W
BINDER_WRITE_READ	struct binder_write_read	Read & Write
BINDER_SET_IDLE_TIMEOUT	int64_t	Write Only
BINDER_SET_MAX_THREADS	size_t	Write Only
BINDER_SET_IDLE_PRIORITY	int	Write Only
BINDER_SET_CONTEXT_MGR	int	Write Only
BINDER_THREAD_EXIT	int	Write Only
BINDER_VERSION	struct binder_version	Read Only

BINDER_WRITE_READ

- Binder의 모든 핵심 기능은 `ioctl` 명령 `BINDER_WRITE_READ`를 통해 제공된다.
- `write_buffer`와 `read_buffer`는 command 배열이다.

```
struct binder_write_read {
            long                write_size;         /* bytes to write */
            long                write_consumed;     /* bytes consumed by driver */
            unsigned long       write_buffer;
            long                read_size;          /* bytes to read */
            long                read_consumed;      /* bytes consumed by driver */
            unsigned long       read_buffer;
};

#include <sys/ioctl.h>
#include <linux/binder.h>

int binder_write(int fd, void *data, long len) {
            struct binder_write_read bwr;

            bwr.write_size = len;
            bwr.write_consumed = 0;
            bwr.write_buffer = (unsigned) data;
            bwr.read_size = 0;
            bwr.read_consumed = 0;
            bwr.read_buffer = 0;
            return ioctl(fd, BINDER_WRITE_READ, &bwr);
}
```



Binder Transaction

- Target Method
  - `handle` : Remote Interface
  - `ptr` & `cookie` : Local Interface
  - `code` : Method ID
- Parcel - Input/Output Parameters
  - `data.ptr.buffer`
  - `data_size`
- Object Reference Management
  - `data.ptr.offsets`
  - `offsets_size`
- Security
  - `sender_pid`
  - `sender_euid`
- No Transaction GUID
  - `Transparent Recursion`

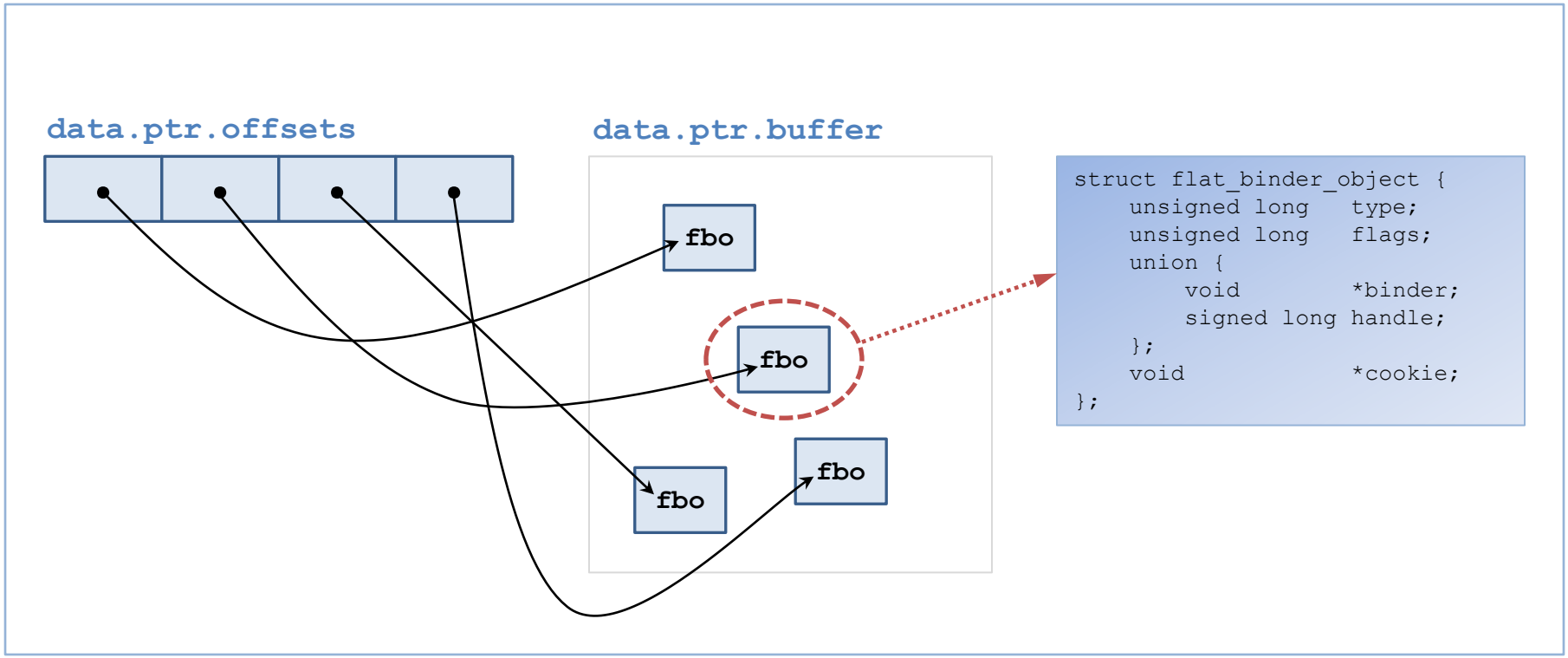
```
#define BC_TRANSACTION
#define BC_REPLY
#define BR_TRANSACTION
#define BR_REPLY

struct binder_transaction_data {
    union {
        size_t          handle;
        void            *ptr;
    } target;
    void                *cookie;
    unsigned int        code;
    unsigned int        flags;
    pid_t               sender_pid;
    uid_t               sender_euid;
    size_t              data_size;
    size_t              offsets_size;
    union {
        struct {
            const void  *buffer;
            const void  *offsets;
        } ptr;
        uint8_t         buf[8];
    } data;
};
```

Standard Parcel

- `data.ptr.buffer` & `data_size`
- 4 bytes alignment with padding
- Left-to-right parameter ordering
- `String16` format - Unicode 문자열을 표현
  - 문자열의 길이를 나타내는 4 bytes 정수로 시작
  - 2 바이트 unicode로 표현된 문자열로 채운 후
  - 끝을 나타내는 2바이트 0으로 마무리.
  - 문자열의 길이가 짝수면 2바이트 padding 0.
- `BC_TRANSACTION`
  - Interface의 이름 - String16
  - Input Parameters
- `BC_REPLY`
  - Return Value
  - Output Parameters

Object Reference Management

- `data.ptr.offsets`는 `data.ptr.buffer`에 저장된 `flat_binder_object` 구조체의 offset을 저장하는 `int` 배열.



Object Reference Management

- Object Reference
  - `BINDER_TYPE_HANDLE`
    - Remote `IBinder`
    - `handle`
  - `BINDER_TYPE_BINDER`
    - Local `IBinder`
    - `binder` & `cookie`
- Binder Driver는 `data.ptr.buffer`에 포함된 `flat_binder_object`들을 자동 변환.
- `binder_transaction_data`의 `target` & `cookie` 역시 `flat_binder_object`.
- `IBinder`
  - Binder Driver의 관점에서 볼 때 `IBinder`가 어떤 Interface 인지는 중요하지 않다.
    Binder Object의 실체를 갖고 있는 프로세스 내에서만 포인터 값이 Type과 연결된 의미를 갖기 때문에, Binder Driver의 입장에서는 `void*` 와 다르지 않다.
  - `IBinder`는 User Space에 존재하는 라이브러리를 통해서만 그 의미를 갖는다.
  - SDK는 Remote(Proxy)와 Local(Stub) 모두 `IBinder` Interface를 통해 노출한다.

Service Manager

- Context Manager
  - Binder Driver의 전역 변수로 하나만 존재.
  - `BINDER_SET_CONTEXT_MGR` – ioctl
  - Handle Zero
  - Booting 시에 Service Manager로 설정됨.

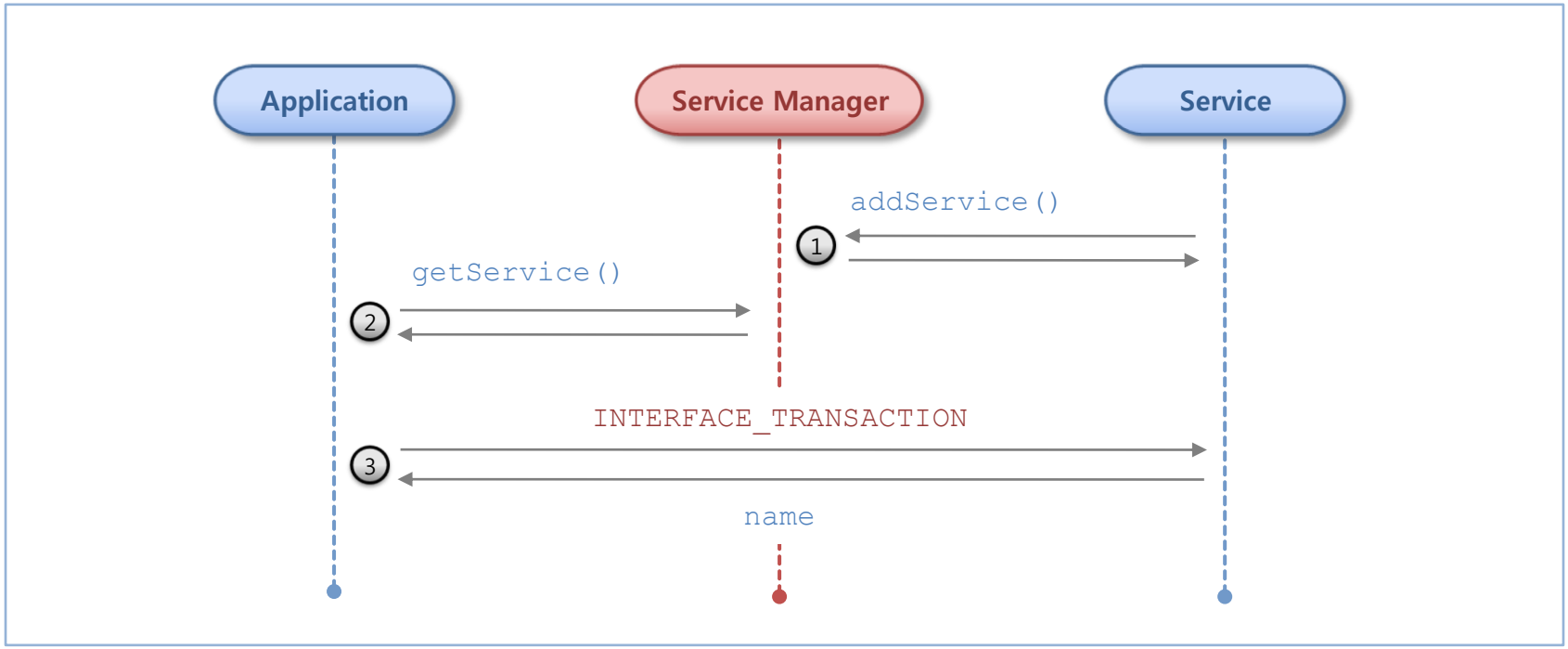
- Service Manager – `android.os.IServiceManager`
  - 응용 프로그램들이 시스템 서비스들을 찾도록 돕는 것을 목적으로 하는 Daemon Process.
  - 부팅 시에 시스템 계정으로 실행되는 데몬 서비스들만 관리.
    - 서비스 등록을 취소하는 함수 없음.
    - 미리 정한 범위에 속하지 않는 요청들은 거부
    - 사용자들의 서비스는 Activity Manager가 관리.
  - /system/bin/service 명령행 도구가 제공됨.
  - PDK 환경 하에서 `BpServiceManager`라는 C++ 클래스로 감싸져서, 좀 더 편리한 Interface로 변형됨.

```
// Native Interface

package android.os;

interface IServiceManager {
    IBinder getService(String name);
    IBinder checkService(String name);
    int addService(String name, IBinder service);
    String listServices(int n);
}
```

```
// PDK Interface of BpServiceManager

package android.os;

interface IServiceManager {
    IBinder getService(String name);
    IBinder checkService(String name);
    int addService(String name, IBinder service);
    List<String> listServices();
}
```

Service Registration & Discovery

- 시스템 서비스들은 실행될 때 `IServiceManager`의 `addService()`를 호출.
- `addService()`의 `service` 파라미터는 Binder Driver에 의해 `handle`로 변환됨.
- Binder Driver는 `name`으로부터 `service`를 찾는 Map을 관리.
- `IServiceManager`의 `getService()`를 통해 다른 프로세스들이 등록된 서비스들의 Interface handle을 찾을 수 있음.
- `android.os.IBinder.INTERFACE_TRANSACTION code`를 사용하여 Interface handle의 실제 이름을 찾는다.



```
# service list
service list
Found 44 services:
0       phone: [com.android.internal.telephony.ITelephony]
1       iphonesubinfo: [com.android.internal.telephony.IPhoneSubInfo]
2       simphonebook: [com.android.internal.telephony.IIccPhoneBook]
3       isms: [com.android.internal.telephony.ISms]
4       appwidget: [com.android.internal.appwidget.IAppWidgetService]
5       audio: [android.media.IAudioService]
6       wallpaper: [android.app.IWallpaperService]
7       checkin: [android.os.ICheckinService]
8       search: [android.app.ISearchManager]
9       location: [android.location.ILocationManager]
10      devicestoragemonitor: []
11      mount: [android.os.IMountService]
12      notification: [android.app.INotificationManager]
13      accessibility: [android.view.accessibility.IAccessibilityManager]
14      connectivity: [android.net.IConnectivityManager]
15      wifi: [android.net.wifi.IWifiManager]
16      netstat: [android.os.INetStatService]
17      input_method: [com.android.internal.view.IInputMethodManager]
18      clipboard: [android.text.IClipboard]
19      statusbar: [android.app.IStatusBar]
20      window: [android.view.IWindowManager]
21      sensor: [android.hardware.ISensorService]
22      alarm: [android.app.IAlarmManager]
23      hardware: [android.os.IHardwareService]
24      battery: []
25      content: [android.content.IContentService]
26      permission: [android.os.IPermissionController]
27      activity.providers: []
28      activity.senders: []
29      activity.services: []
30      activity.broadcasts: []
31      cpuinfo: []
32      meminfo: []
33      activity: [android.app.IActivityManager]
34      package: [android.content.pm.IPackageManager]
35      telephony.registry: [com.android.internal.telephony.ITelephonyRegistry]
36      usagestats: [com.android.internal.app.IUsageStats]
37      batteryinfo: [com.android.internal.app.IBatteryStats]
38      power: [android.os.IPowerManager]
39      entropy: []
40      SurfaceFlinger: [android.ui.ISurfaceComposer]
41      media.camera: [android.hardware.ICameraService]
42      media.player: [android.hardware.IMediaPlayerService]
43      media.audio_flinger: [android.media.IAudioFlinger]
#
```

```
# service call activity 1598968902
service call activity 1598968902
Result: Parcel(
  0x00000000: 0000001c 006e0061 00720064 0069006f '....a.n.d.r.o.i.'
  0x00000010: 002e0064 00700061 002e0070 00410049 'd...a.p.p...I.A.'
  0x00000020: 00740063 00760069 00740069 004d0079 'c.t.i.v.i.t.y.M.'
  0x00000030: 006e0061 00670061 00720065 00000000 'a.n.a.g.e.r.....')
#
```

바인더의 한계와 전망

- NDK 지원의 부재
  - <linux/binder.h>는 unstable
  - AIDL과 Binder Library는 C++을 지원하지 않음.
  - PDK에는 부분적인 C++ 구현이 존재.
- Binder Shell?
  - Scripting에 대한 전망의 부재
  - Introspection – AIDL Registry?
- No Exception
  - 정말로 필요한가?
  - Error는 IPC 수준에서 지원하고 있음.
  - Library 수준에서 구현할 수도 있음.
  - Parcelable Exception & Extended AIDL
  - NDK가 C++ Exception을 지원하지 않는다는 것을 기억하는가?
- APC & Cancellation?
  - CCR-like Feature Set?
  - Thread Migration Model과의 충돌이 있을 가능성이 있음.
- Networking
  - RPC over TCP or UDP
  - Phone-to-Phone or Phone-to-Server RPC
  - Endian Conversion
  - Low-latency Multichannel Reliable Transport

감사합니다

남은 질문과 질책은

www.flowdas.com
bookman@flowdas.com

으로 보내주세요