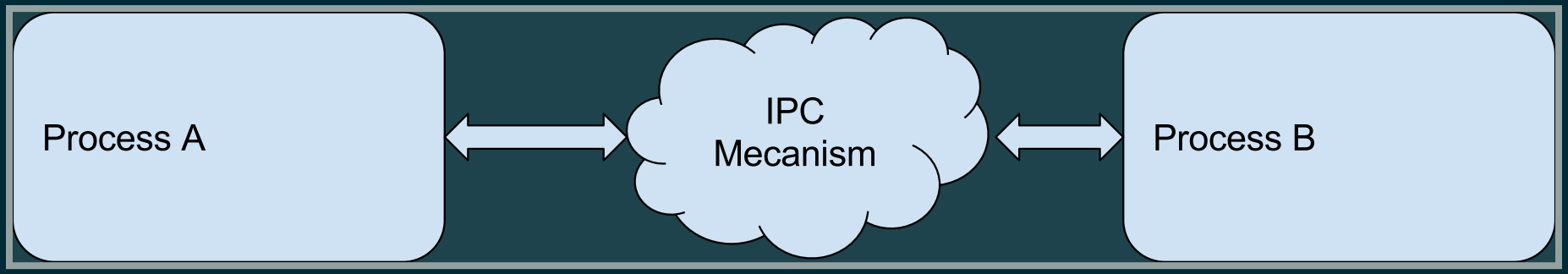
BINDER

THE ANDROID IPC FRAMEWORK

Antoine 'xdbob' Damhet

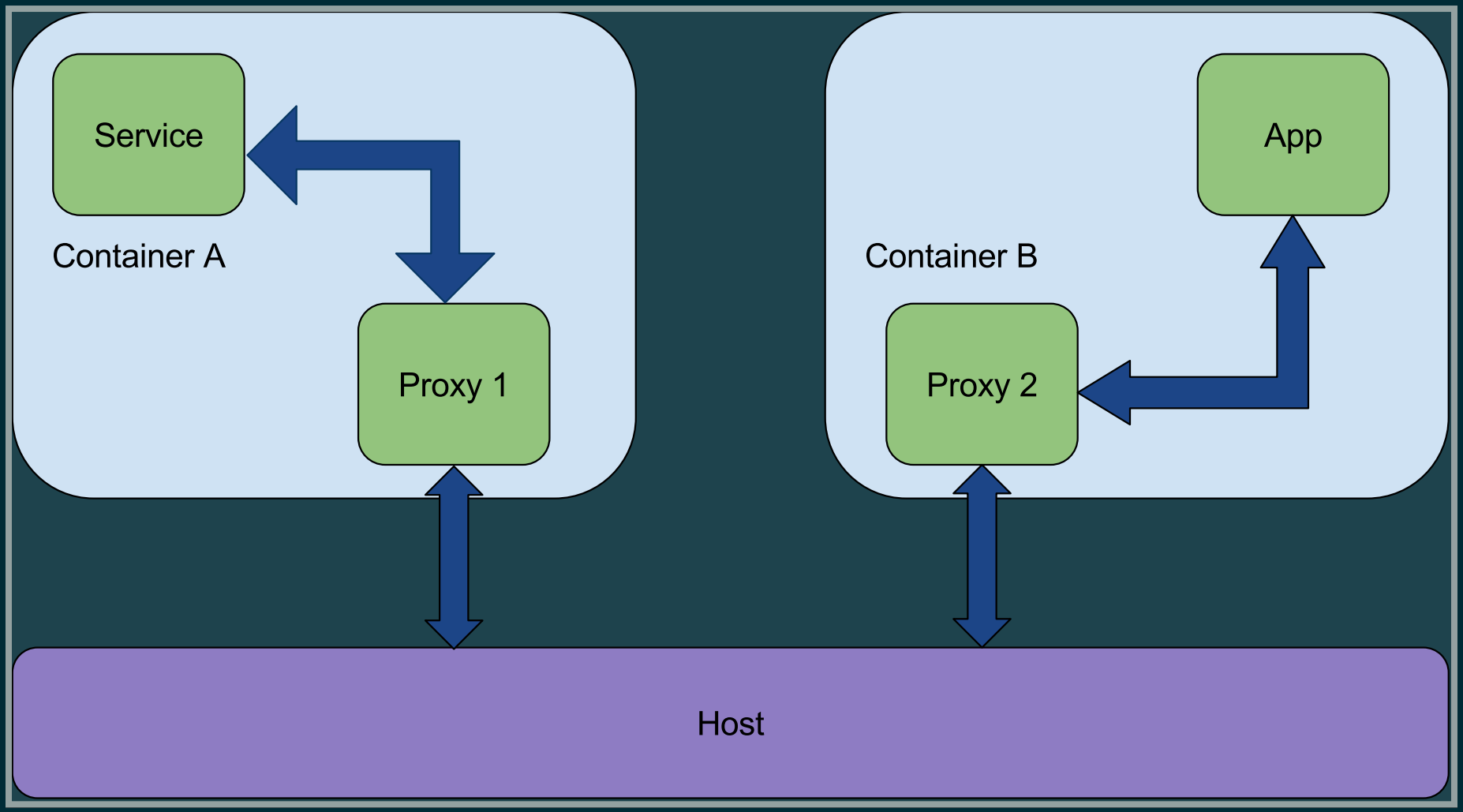
July 15, 2016

THE INTERPROCESS COMMUNICATION



- Share memory
- Share privileges
- Call a remote functions (RPC)
- Synchronize the processes

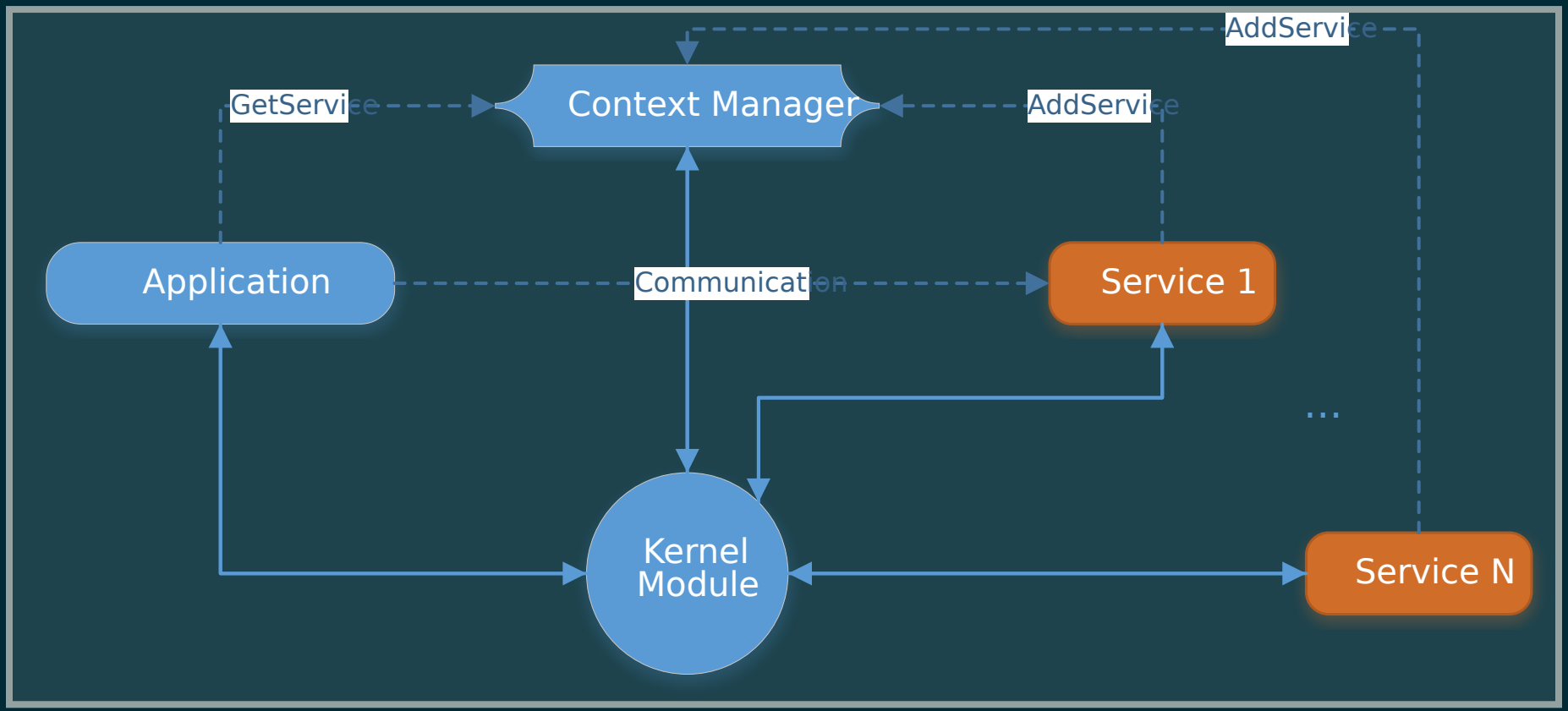
THE GOAL



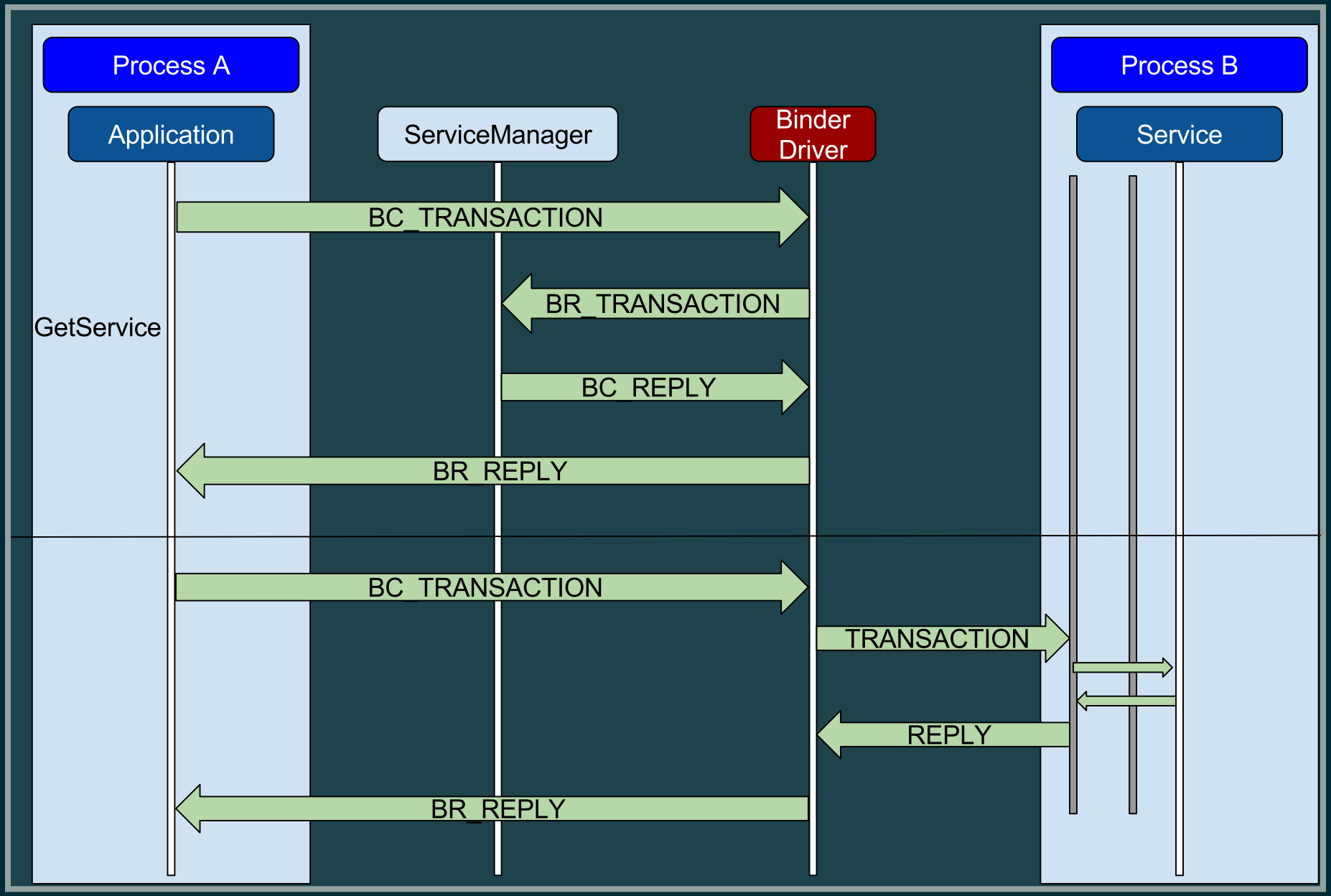
OPENBINDER TO ANDROID'S BINDER

- Developped by Be Inc then Palm, Inc.
- Running on BeOS, Windows CE and PalmOS Cobalt
- 2001 => 2006
- ~2008
- Fork/rewrite of OpenBinder due to a license incompatibility

HOW BINDER WORKS ?



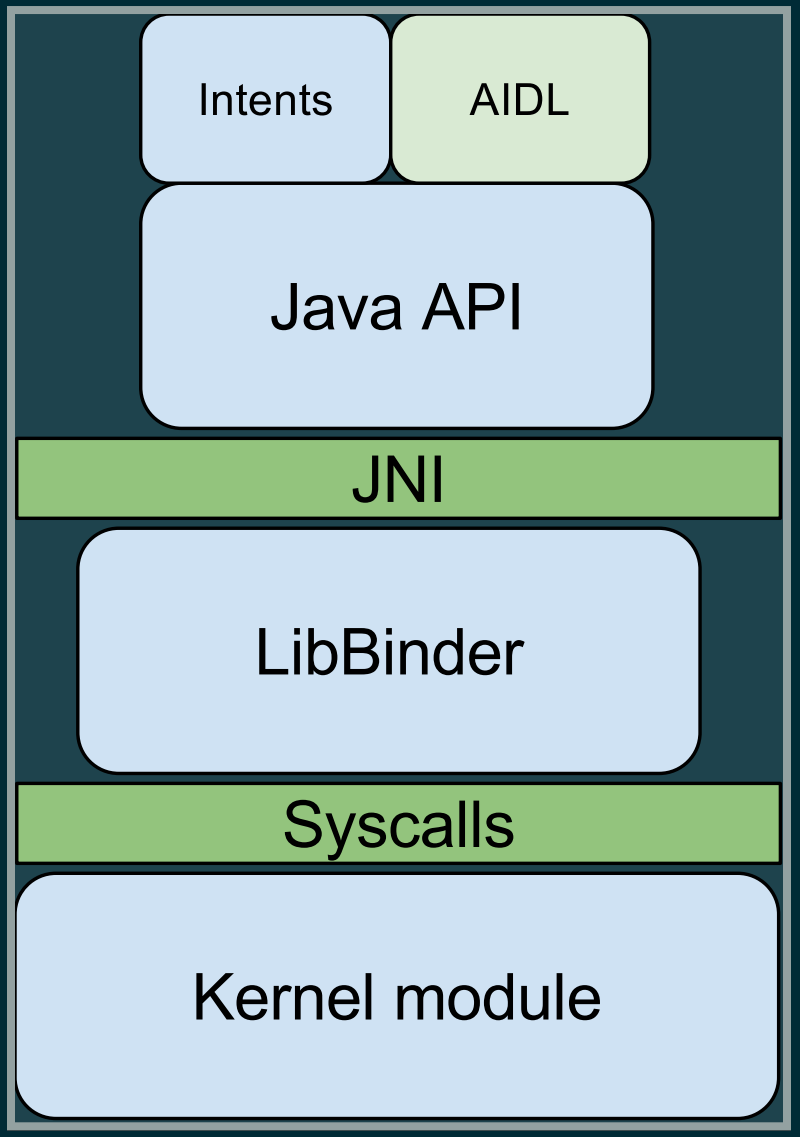
EXAMPLE OF AN INTERACTION BETWEEN A SERVICE AND AN APP



BINDER CAPABILITIES

- Share memory
  - The memory is allocated via ashmem
  - It's shared via a file descriptor in Binder
- Link to death
- Remote Procedure Call (RPC)
  - One way
  - Two way
- Get indentifications
  - Get UID
  - Get PID
- Maps objects across processes
- Reference Counting

BINDER STACK



JAVA

INTENTS

- Action
- Data

```
{ACTION_DIAL, "tel:123"}
```

- Implicit
- Explicit

AIDL

(ANDROID INTERFACE DEFINITION LANGUAGE)

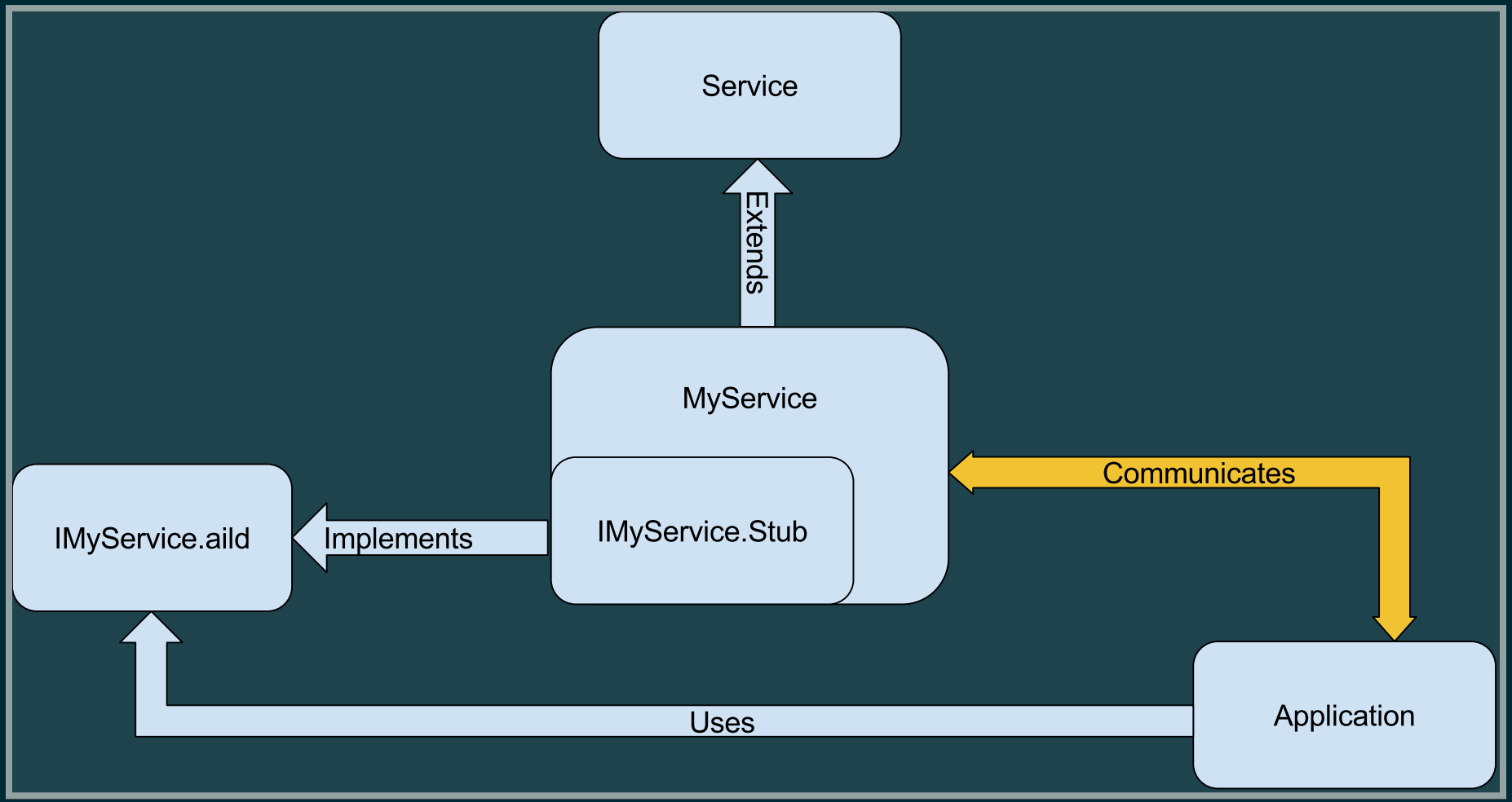
AIDL EXAMPLE

```
package com.example.android;

/** Example service interface */
interface IRemoteService {
    /**
    * Request the process ID of this service,
    * to do evil things with it.
    **/
    int getPid();

    /**
    * Demonstrates some basic types that you can use as parameters
    * and return values in AIDL.
    **/
    void basicTypes(int anInt, long aLong, boolean aBoolean,
        float aFloat, double aDouble, String aString);
}
```

AIDL


Service
Extends
MyService
IMyService.aild
Implements
IMyService.Stub
Communicates
Application
Uses

LIBBINDER (C++)

- Parcel
- Marshalling/unmarshalling
- Uses syscalls to communicate with the kernel driver

KERNEL MODULE

strace a process to see what happens

```
[...]
openat(AT_FDCWD, "/dev/binder", O_RDWR|O_LARGEFILE) = 3
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
ioctl(3, BINDER_VERSION, 0xbee72938)    = 0
ioctl(3, BINDER_SET_MAX_THREADS, 0xbee7293c) = 0
mmap2(NULL, 1040384, PROT_READ, MAP_PRIVATE|MAP_NORESERVE, 3, 0) = 0xb6b82000
[...]
ioctl(3, _IOC(_IOC_READ|_IOC_WRITE, 0x62, 0x01, 0x18), 0xbee72808) = 0
ioctl(3, _IOC(_IOC_READ|_IOC_WRITE, 0x62, 0x01, 0x18), 0xbee72808) = 0
ioctl(3, _IOC(_IOC_READ|_IOC_WRITE, 0x62, 0x01, 0x18), 0xbee72800) = 0
ioctl(3, _IOC(_IOC_READ|_IOC_WRITE, 0x62, 0x01, 0x18), 0xbee72800) = 0
[...]
```

```
strace -v service check phone
```

WHAT DOES THE KERNEL MODULE EXPORT ?

```
static const struct file_operations binder_fops = {
    .owner = THIS_MODULE,
    .poll = binder_poll,
    .unlocked_ioctl = binder_ioctl,
    .compat_ioctl = binder_ioctl,
    .mmap = binder_mmap,
    .open = binder_open,
    .flush = binder_flush,
    .release = binder_release,
};
```

MMAP(2) ?

```
void *mmap(void *addr, size_t length, int prot, int flags,
       int fd, off_t offset);
```

PROT_WRITE forbidden

```
ProcessState::ProcessState()
{
[...]
    // mmap the binder, providing a chunk of virtual address space to receive transa
    mVMStart = mmap(0, BINDER_VM_SIZE, PROT_READ, MAP_PRIVATE | MAP_NORESERVE, mDriv
[...]
}
```

frameworks/native/libs/binder/ProcessState.cpp

LOOKING AT THE IOCTL'S

WHAT IS IOCTL(2)

In computing, ioctl (an abbreviation of input/output control) is a system call for device-specific input/output operations and other operations which cannot be expressed by regular system calls.

-- Wikipedia

```
int ioctl(int fildes, int request, ... /* arg */);
```

AFTER A QUICK LOOK AT 'BINDER.H'

```
#ifdef BINDER_IPC_32BIT
typedef __u32 binder_size_t;
typedef __u32 binder_uintptr_t;
#else
typedef __u64 binder_size_t;
typedef __u64 binder_uintptr_t;
#endif
```

```
#define BINDER_WRITE_READ         _IOWR('b', 1, struct binder_write_read)
#define BINDER_SET_IDLE_TIMEOUT   _IOW('b', 3, __s64)
#define BINDER_SET_MAX_THREADS    _IOW('b', 5, __u32)
#define BINDER_SET_IDLE_PRIORITY  _IOW('b', 6, __s32)
#define BINDER_SET_CONTEXT_MGR    _IOW('b', 7, __s32)
#define BINDER_THREAD_EXIT        _IOW('b', 8, __s32)
#define BINDER_VERSION            _IOWR('b', 9, struct binder_version)
```

IS IT BINDER_WRITE_READ ?

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

ANY OTHER IOCTLS ?

```
enum binder_driver_return_protocol {
    BR_ERROR = _IOR('r', 0, __s32),
    BR_OK = _IO('r', 1),
    BR_TRANSACTION = _IOR('r', 2, struct binder_transaction_data),
    BR_REPLY = _IOR('r', 3, struct binder_transaction_data),
[...]
    BR_FAILED_REPLY = _IO('r', 17),
};
```

```
enum binder_driver_command_protocol {
    BC_TRANSACTION = _IOW('c', 0, struct binder_transaction_data),
    BC_REPLY = _IOW('c', 1, struct binder_transaction_data),
[...]
    BC_DEAD_BINDER_DONE = _IOW('c', 16, binder_uintptr_t),
};
```

TRANSACTION/REPLY

```
struct binder_transaction_data {
    /* The first two are only used for bcTRANSACTION and brTRANSACTI
     * identifying the target and contents of the transaction.
     */
    union {
        /* target descriptor of command transaction */
        __u32    handle;
        /* target descriptor of return transaction */
        binder_uintptr_t ptr;
    } target;
    binder_uintptr_t    cookie;    /* target object cookie */
    __u32        code;        /* transaction command */

    /* General information about the transaction. */
    __u32            flags;
    pid_t        sender_pid;
    uid_t        sender_euid;
[...]
};
```

```
struct binder_transaction_data {
[...]
    binder_size_t    data_size;    /* number of bytes of data */
    binder_size_t    offsets_size;    /* number of bytes of offsets

    /* If this transaction is inline, the data immediately
     * follows here; otherwise, it ends with a pointer to
     * the data buffer.
     */
    union {
        struct {
            /* transaction data */
            binder_uintptr_t    buffer;
            /* offsets from buffer to flat_binder_object structs */
            binder_uintptr_t    offsets;
        } ptr;
        __u8    buf[8];
    } data;
};
```

DECODING BINDER'S IOCTL

```
openat(AT_FDCWD, "/dev/binder", O_RDWR|O_LARGEFILE) = 3
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
ioctl(3, BINDER_VERSION, {protocol_version=7}) = 0
ioctl(3, BINDER_SET_MAX_THREADS, [15])  = 0
mmap2(NULL, 1040384, PROT_READ, MAP_PRIVATE|MAP_NORESERVE, 3, 0) = 0xb6b02000
ioctl(3, BINDER_WRITE_READ, {write_size=44, write_consumed=0, write_buffer=[{BC_TRA
ioctl(3, BINDER_WRITE_READ, {write_size=0, write_consumed=0, write_buffer=0xb6c2710
ioctl(3, BINDER_WRITE_READ, {write_size=68, write_consumed=0, write_buffer=[{BC_FRE
ioctl(3, BINDER_WRITE_READ, {write_size=0, write_consumed=0, write_buffer=0xb6c2710
```

```
strace -v service check phone
```

BC_TRANSACTION

```
{handle=0}
cookie=0x0
code=2
flags=TF_ACCEPT_FDS
sender_pid=0
sender_euid=0
data_size=80
offsets_size=0
data=["\0\0@\0\32\0\0\0a\0n\0d\0r\0o\0i\0d\0.\0o\0s\0.\0I\0"...
```

BR_REPLY

```
cookie=0x0
code=0
flags=0
sender_pid=0
sender_euid=1000
data_size=16
offsets_size=4
data=[[{
    type=BINDER_TYPE_HANDLE
    flags=FLAT_BINDER_FLAG_ACCEPTS_FDS|7f
    {handle=1}
    cookie=0x0
}]]
```

CONCLUSION

SOURCES

IN LINUX:

- drivers/android/binder.c
- include/uapi/linux/android/binder.h

IN AOSP:

- frameworks/native/libs/binder/Binder.cpp
- frameworks/native/libs/binder/BpBinder.cpp
- frameworks/native/libs/binder/ProcessState.cpp
- frameworks/native/libs/binder/tests/binderDriverInterfaceTest.cpp
- frameworks/native/include/binder/IBinder.h
- frameworks/native/include/binder/Binder.h

IN THE ANDROID DOCUMENTATION:

- https://developer.android.com/reference/android/os/IBinder.html
- https://developer.android.com/reference/android/os/Binder.html

CONTACT

- xdbob@lse.epita.fr
- xdbob on freenode and rezosup and twitter