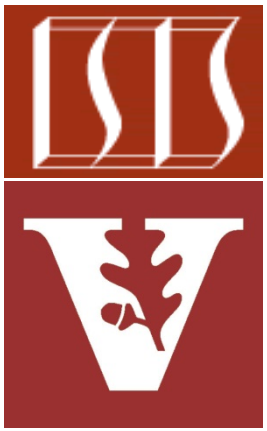
Android Services & Local IPC: The Proxy Pattern (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



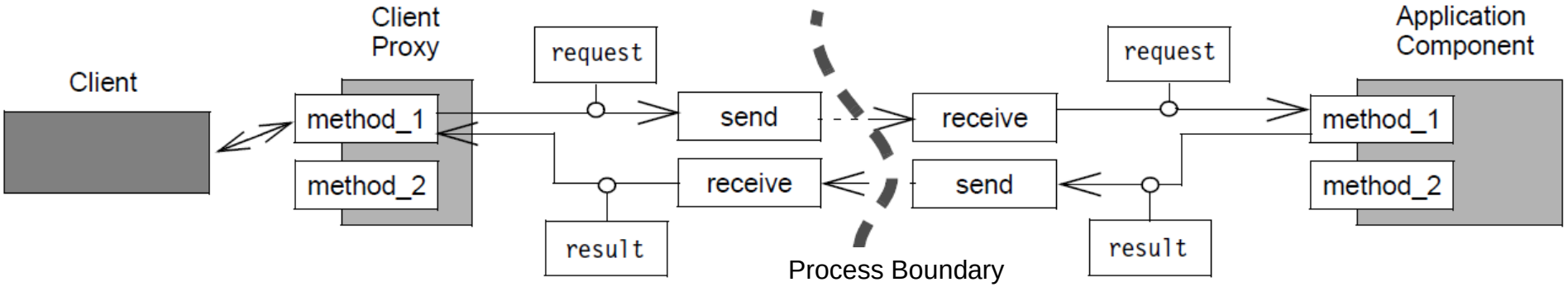
Professor of Computer Science

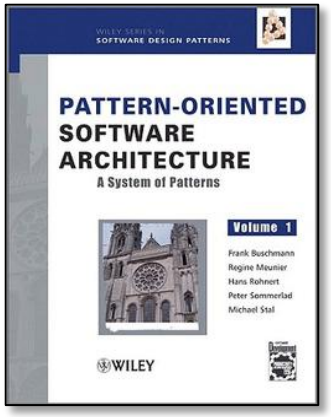
Institute for Software Integrated Systems

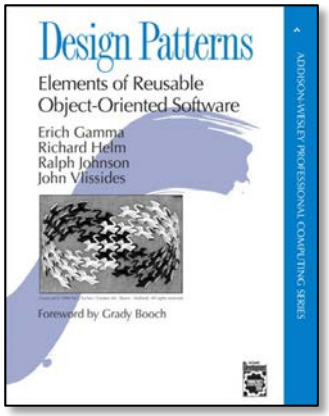
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the *Proxy* pattern





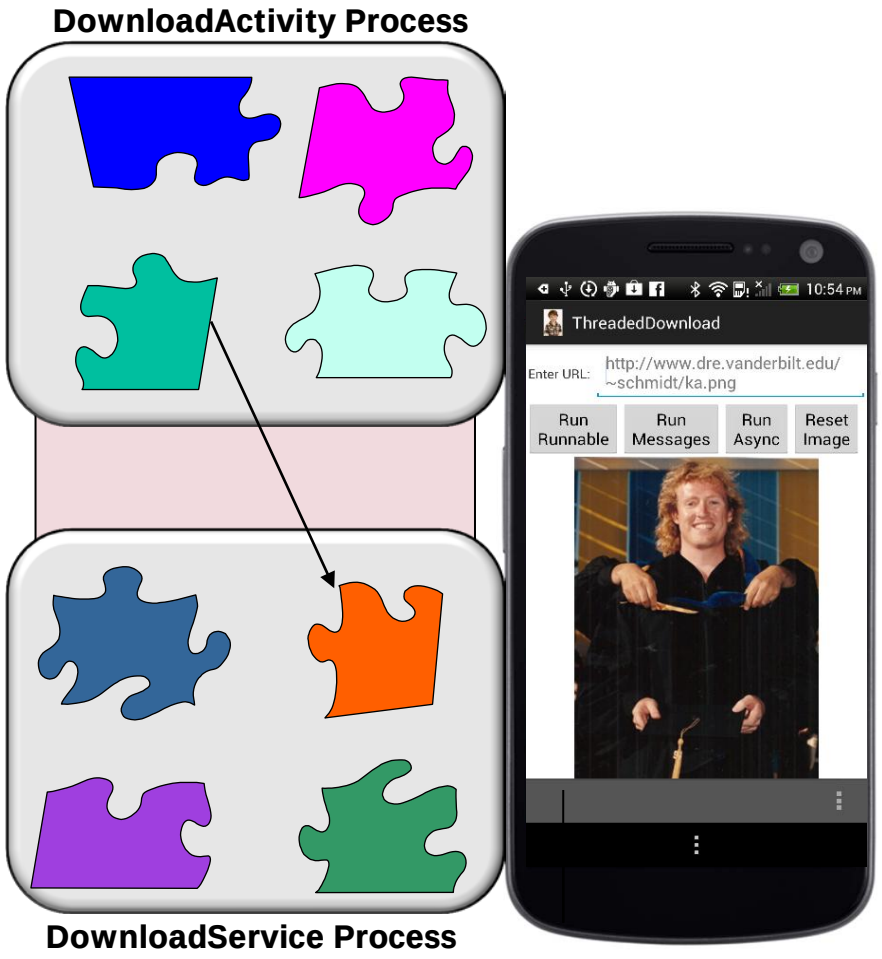


See en.wikipedia.org/wiki/Proxy_pattern for more on *Proxy* pattern

Challenge: Simplifying Access to Remote Objects

Context

- It is often infeasible—or impossible—to access an object directly
  - e.g., may reside in server process

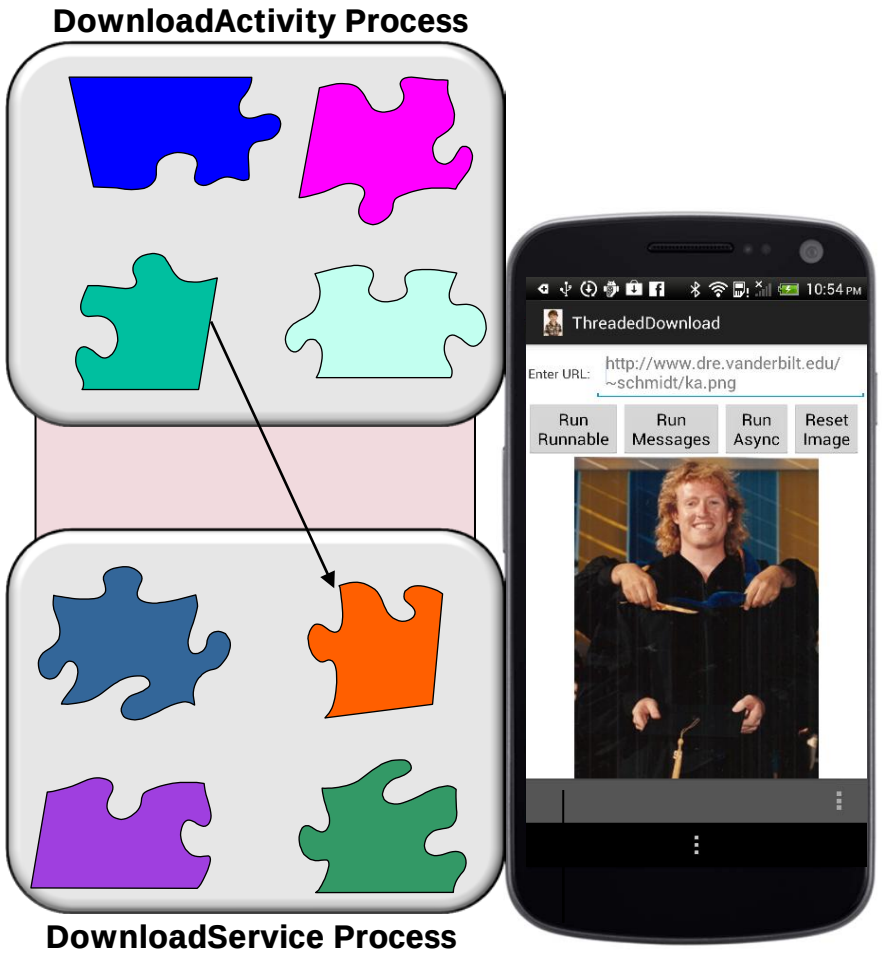


Android's Binder provides a high-performance IPC mechanism

Challenge: Simplifying Access to Remote Objects

Context

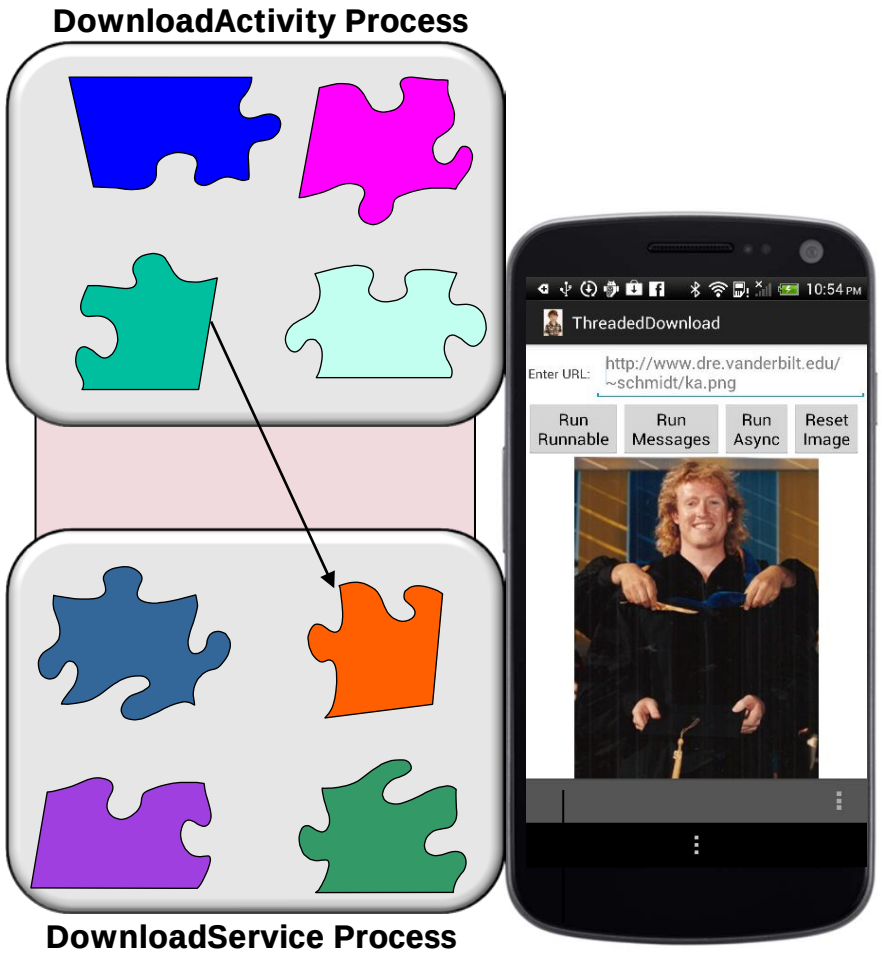
- It is often infeasible—or impossible—to access an object directly
- Partitioning of objects in a system may change as requirements evolve



Challenge: Simplifying Access to Remote Objects

Problems

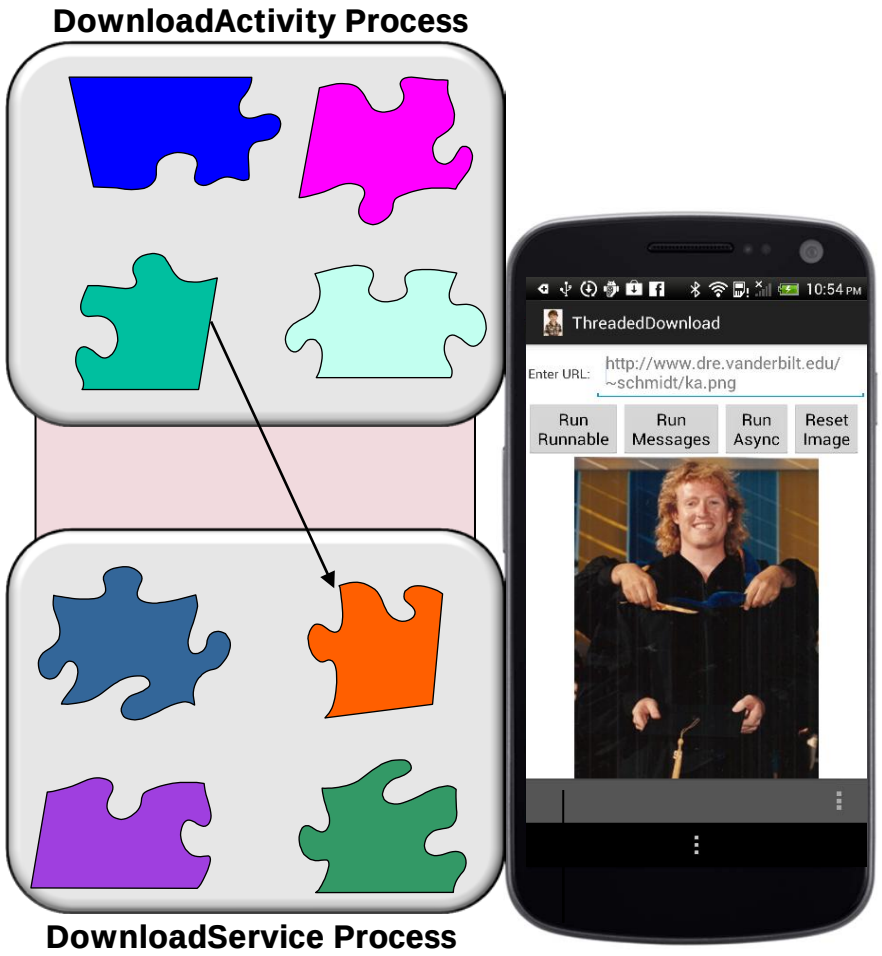
- Manually (de)marshaling messages can be tedious, error-prone, & inefficient



Challenge: Simplifying Access to Remote Objects

Problems

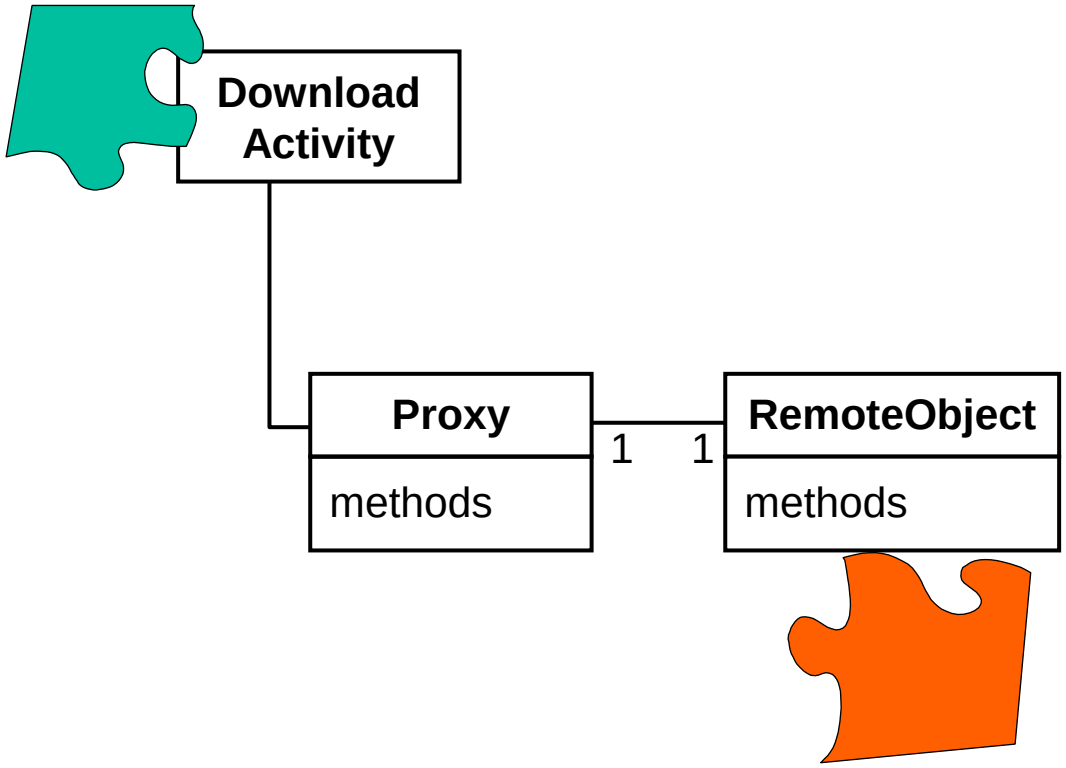
- Manually (de)marshaling messages can be tedious, error-prone, & inefficient
- It is time-consuming to re-write, re-configure, & re-deploy components across address spaces as requirements & environments change



Challenge: Simplifying Access to Remote Objects

Solution

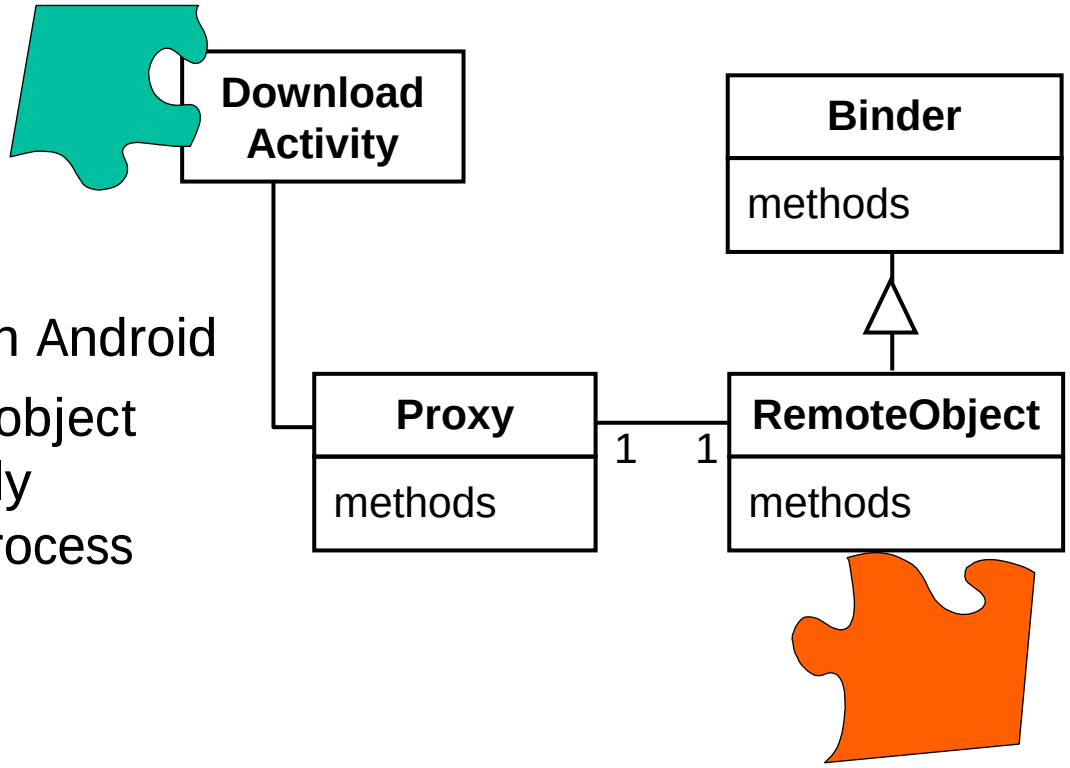
- Define a *proxy* that provides a surrogate thru which clients can access remote objects



Challenge: Simplifying Access to Remote Objects

Solution

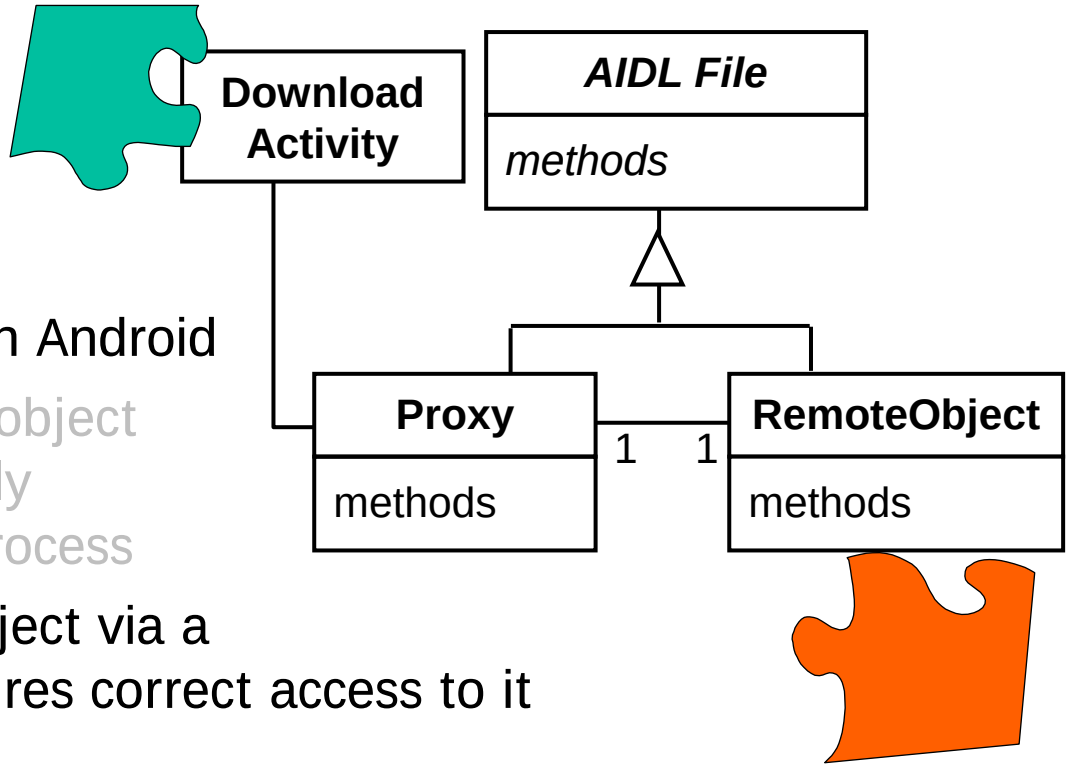
- Define a *proxy* that provides a surrogate thru which clients can access remote objects
- e.g., one way to implement this in Android
  - A service implements a Binder object that a client can't access directly since it may be in a different process



Challenge: Simplifying Access to Remote Objects

Solution

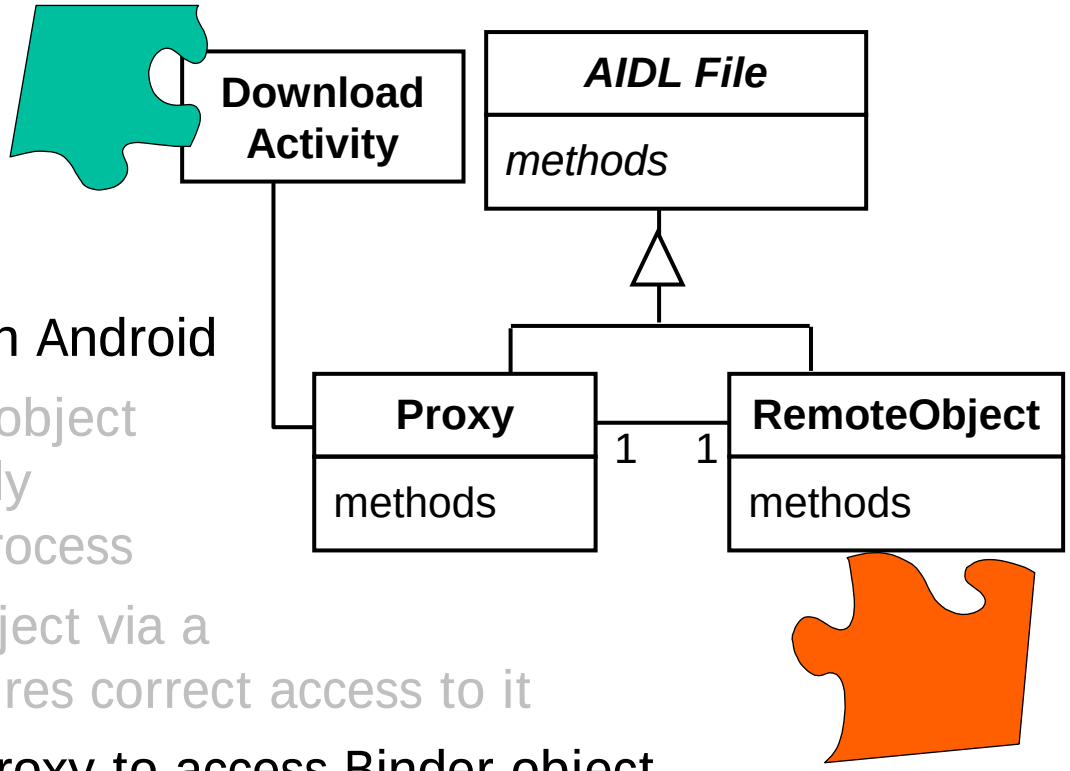
- Define a *proxy* that provides a surrogate thru which clients can access remote objects
- e.g., one way to implement this in Android
  - A service implements a Binder object that a client can't access directly since it may be in a different process
  - Proxy represents the Binder object via a common AIDL interface & ensures correct access to it



Challenge: Simplifying Access to Remote Objects

Solution

- Define a *proxy* that provides a surrogate thru which clients can access remote objects
- e.g., one way to implement this in Android
  - A service implements a Binder object that a client can't access directly since it may be in a different process
  - Proxy represents the Binder object via a common AIDL interface & ensures correct access to it
  - Clients calls a method on the proxy to access Binder object
    - Whether the object is in-process or out-of-process can be controlled via the AndroidManifest.xml config file



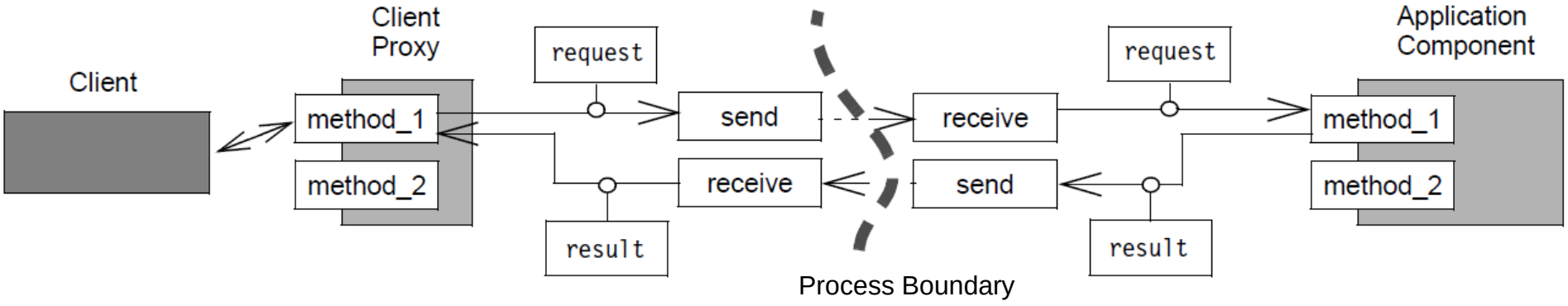
The *Proxy* works together with Binder RPC to implement the *Broker* pattern

Proxy GoF Object Structural

Intent

POSA1 also contains the *Proxy* pattern

- Provide a surrogate or placeholder for another object to control access to it



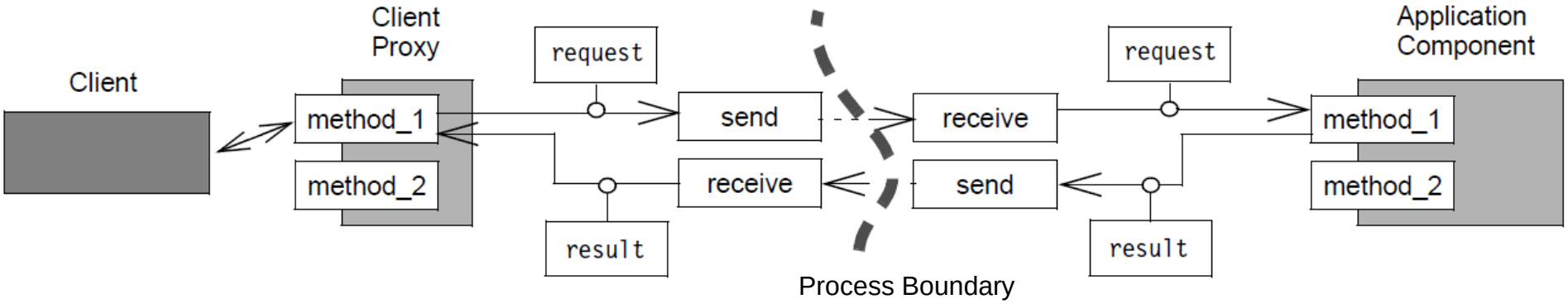
See en.wikipedia.org/wiki/Proxy_pattern for more on *Proxy* pattern

Proxy GoF Object Structural

Applicability

POSA1 also contains the *Proxy* pattern

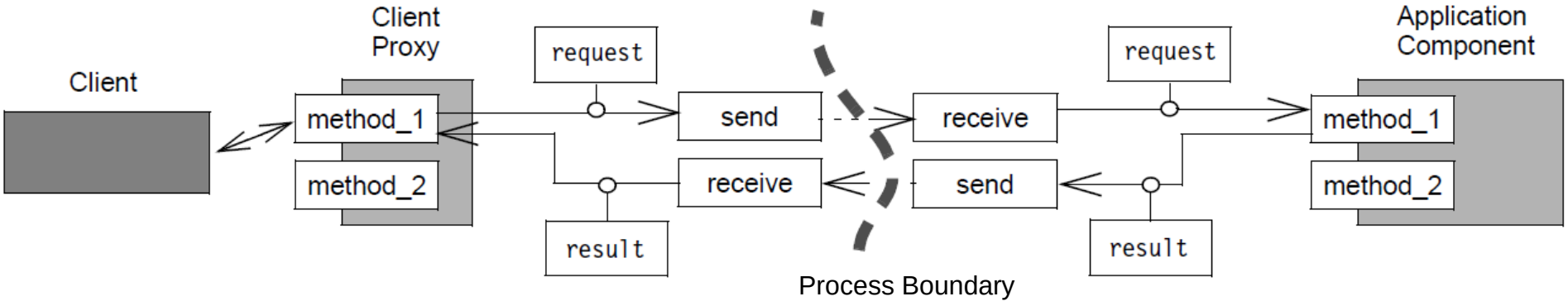
- When there is a need for a more sophisticated reference to a object than a simple pointer or simple reference can provide



Proxy GoF Object Structural

Applicability

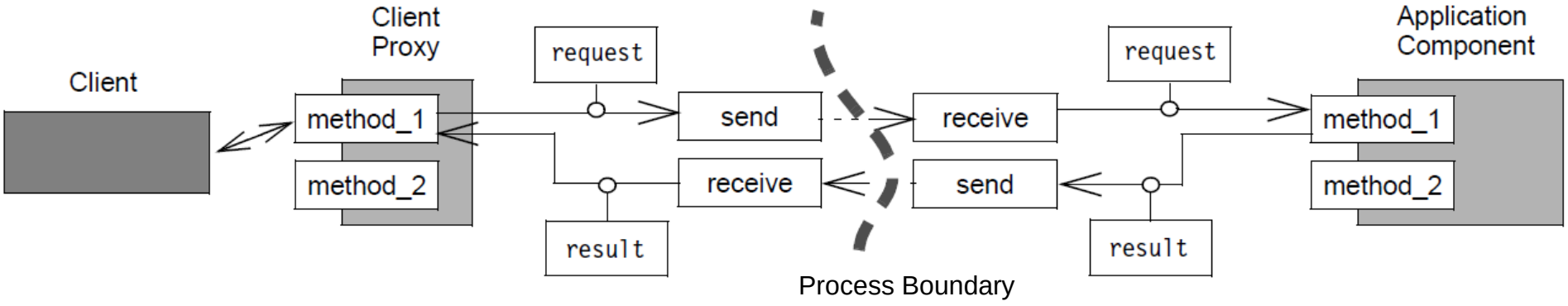
- When there is a need for a more sophisticated reference to a object than a simple pointer or simple reference can provide
- Help ensure remote objects look/act as much like local components as possible from a client app perspective



Proxy GoF Object Structural

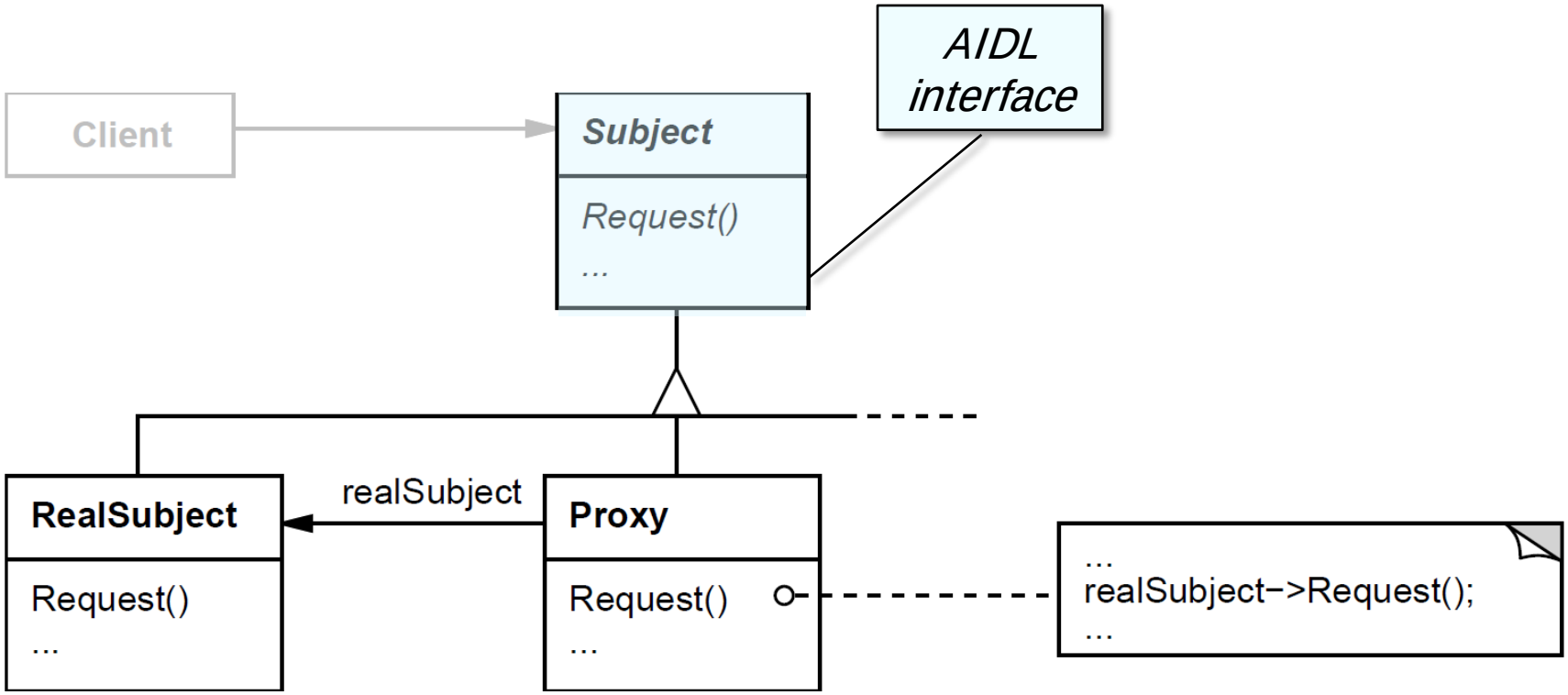
Applicability

- When there is a need for a more sophisticated reference to a object than a simple pointer or simple reference can provide
- Help ensure remote objects look/act as much like local components as possible from a client app perspective
- When there's a need for statically-typed method invocations



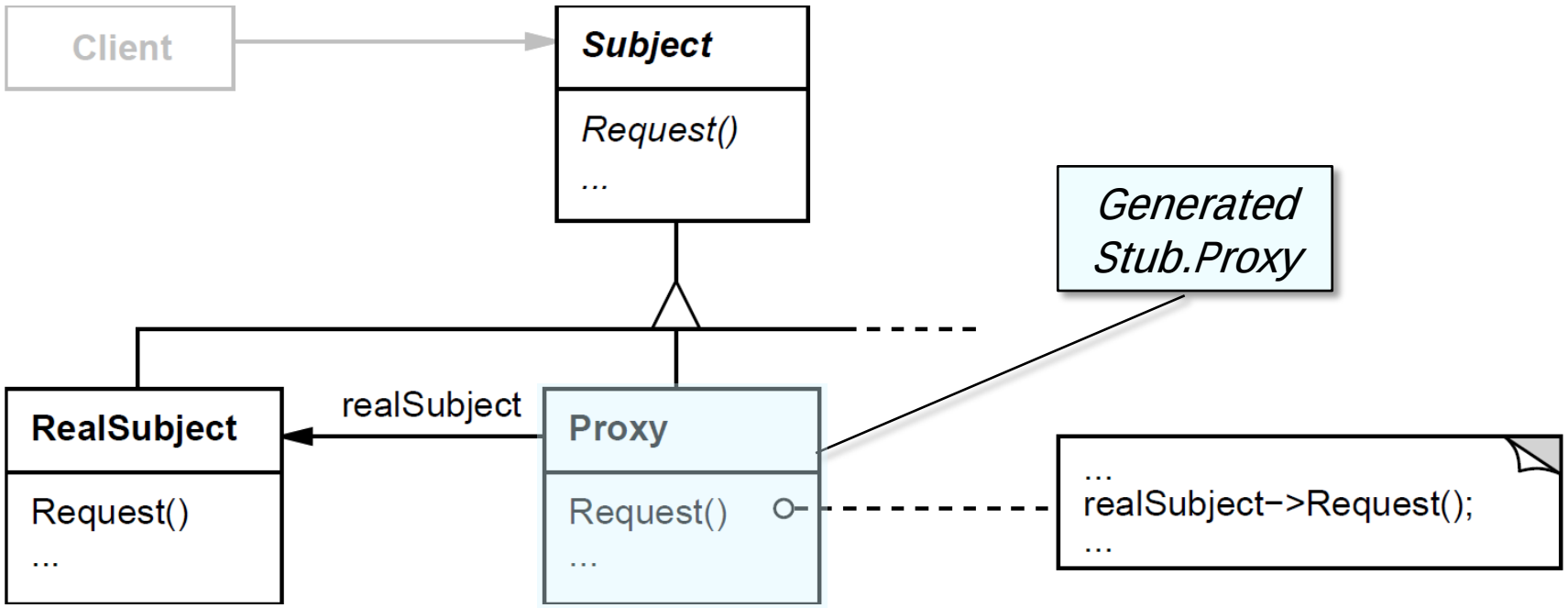
Proxy GoF Object Structural

Structure & Participants



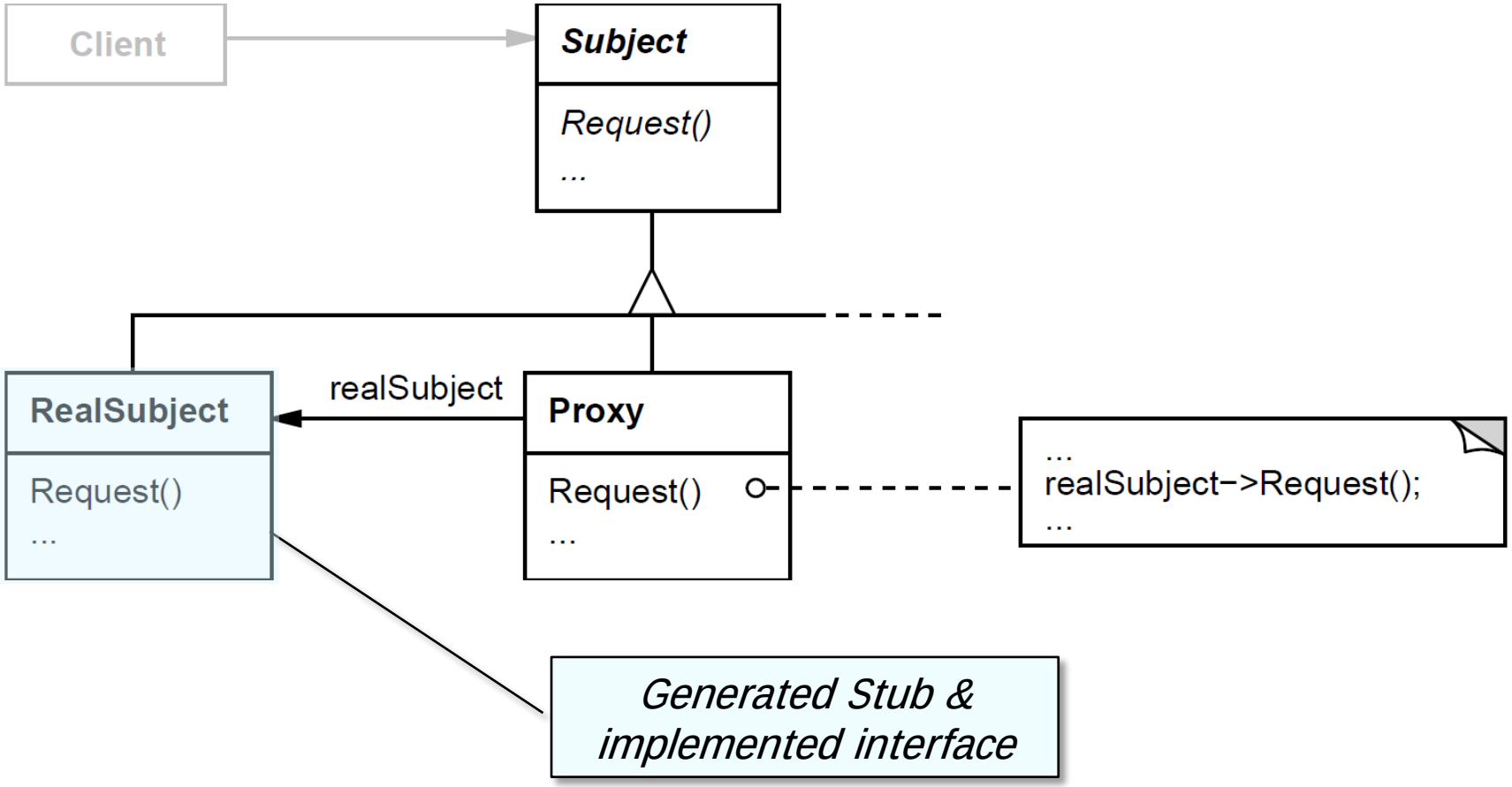
Proxy GoF Object Structural

Structure & Participants



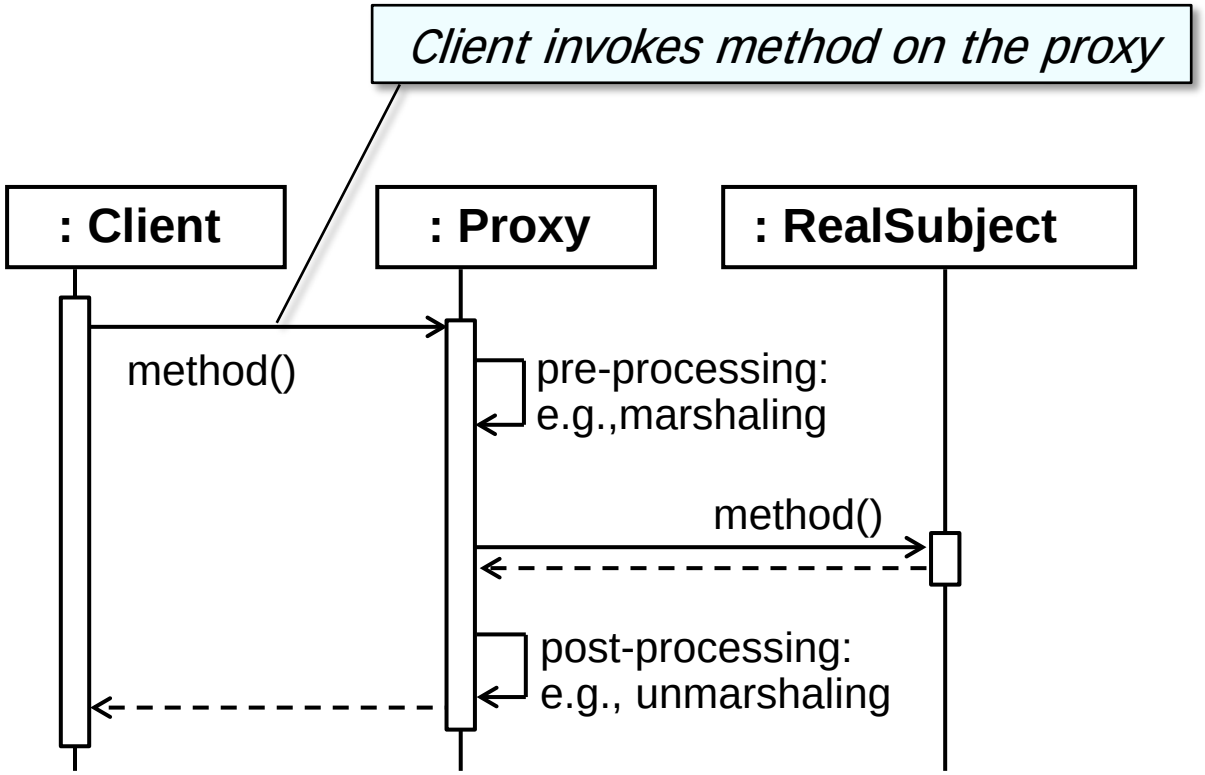
Proxy GoF Object Structural

Structure & Participants



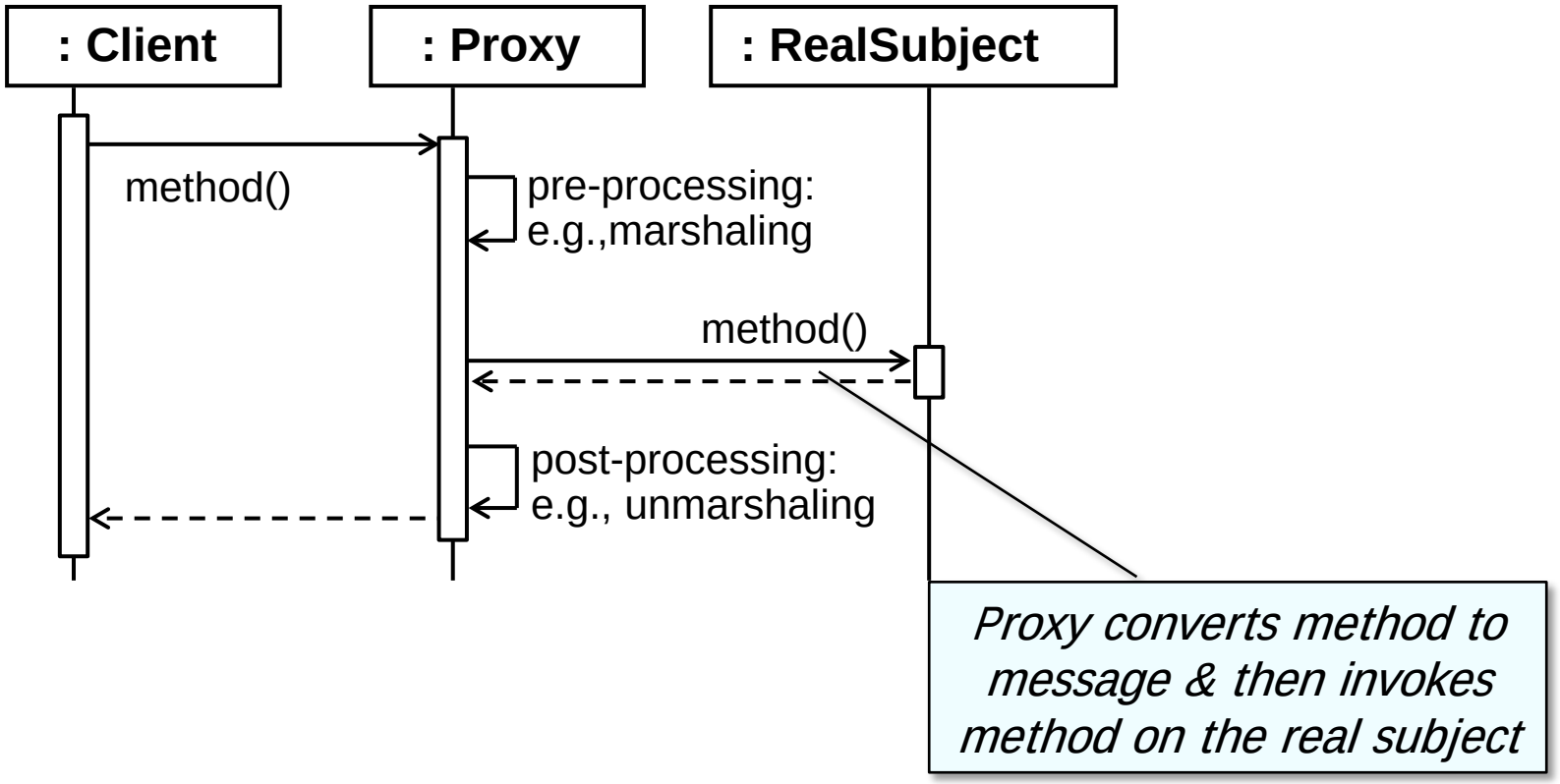
Proxy GoF Object Structural

Dynamics



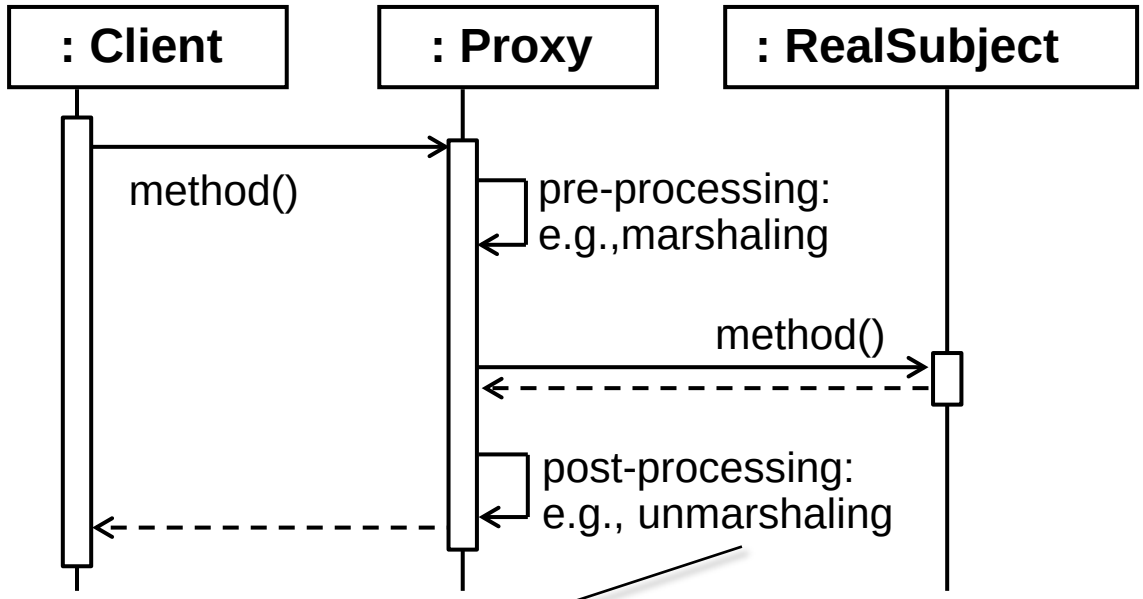
Proxy GoF Object Structural

Dynamics



Proxy GoF Object Structural

Dynamics

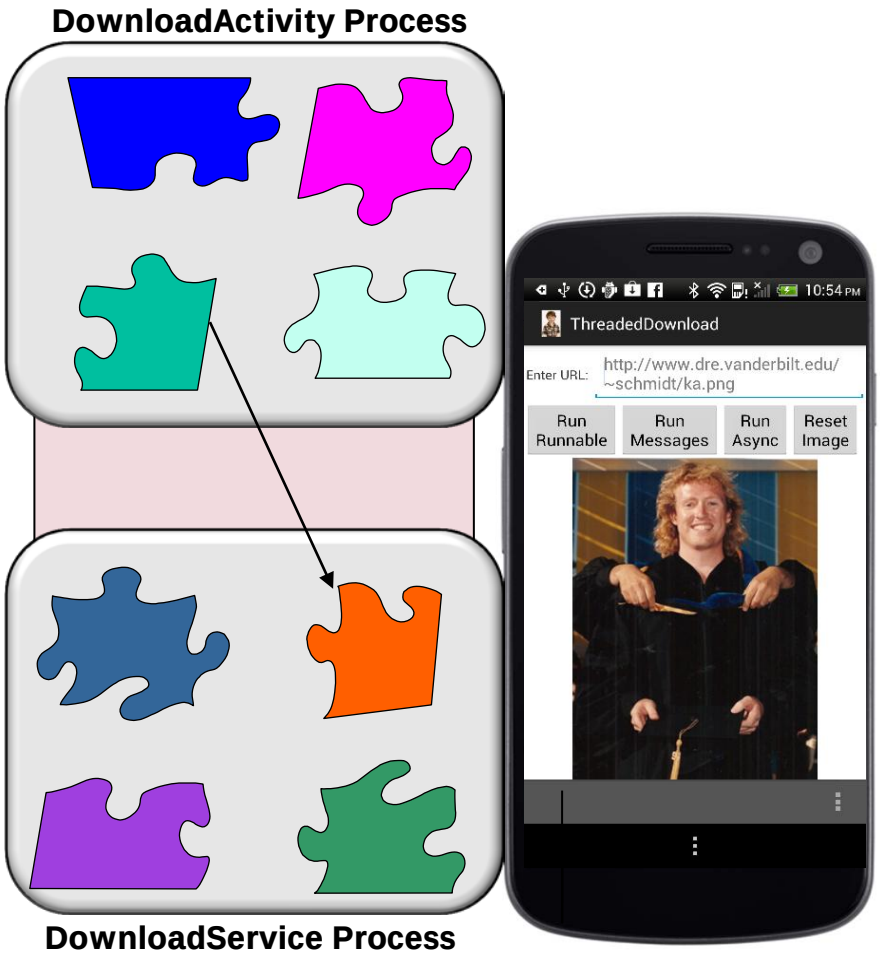


Proxy converts the return from the method call on the RealSubject back into results from the original method

Proxy GoF Object Structural

Consequences

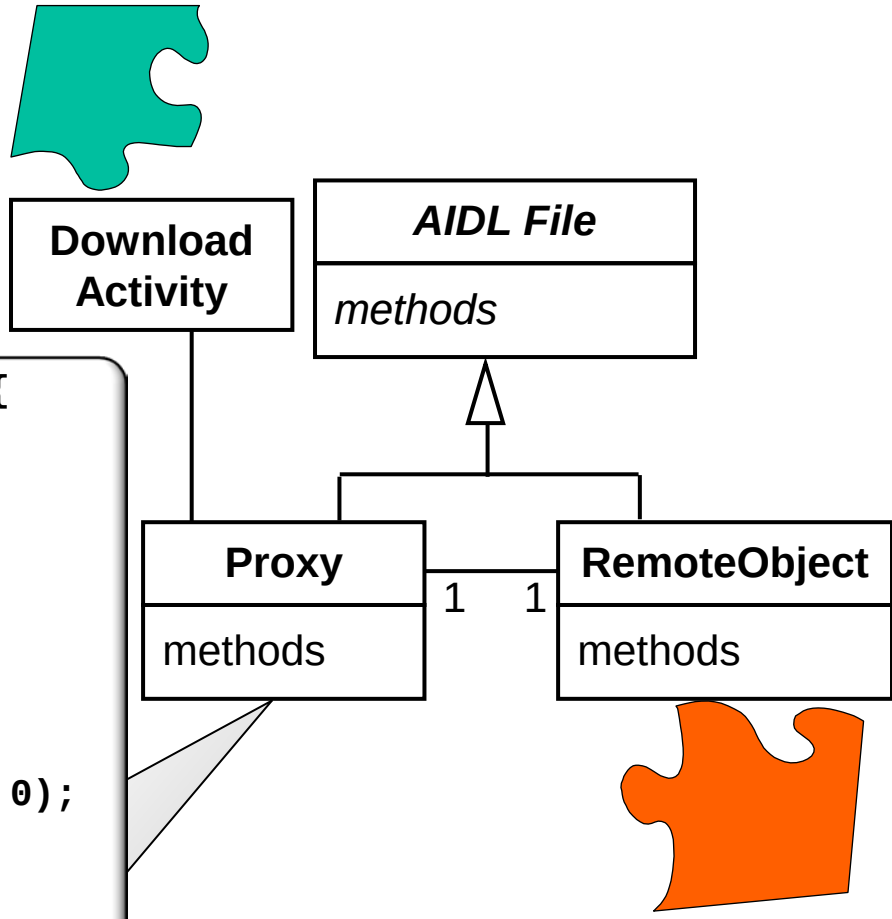
+ Decoupling client from object location



Proxy — GoF Object Structural

Consequences

+ Decoupling client from object location
+ Simplify tedious & error-prone details

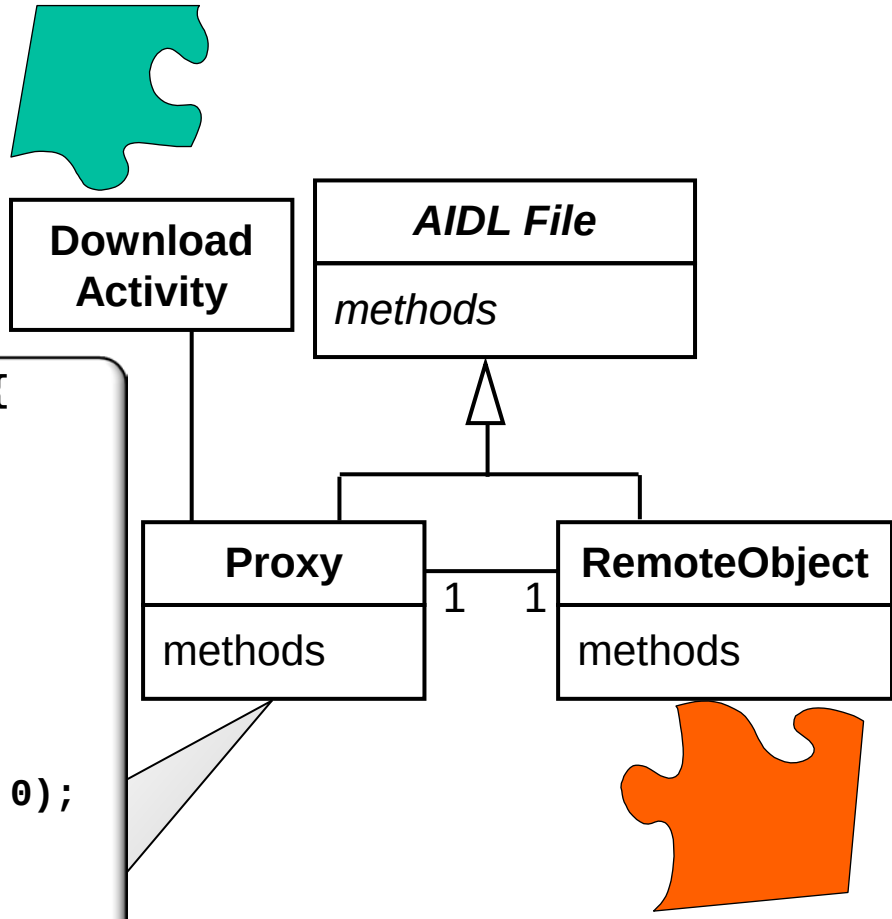



```
public String downloadImage(String uri) ... {
  android.os.Parcel _data =
    android.os.Parcel.obtain();
  android.os.Parcel _reply =
    android.os.Parcel.obtain();
  java.lang.String _result;
  _data.writeInterfaceToken(DESCRIPTOR);
  _data.writeString(uri);
  mRemote.transact(Stub.
   TRANSACTION_downloadImage, _data, _reply, 0);
  _reply.readException();
  _result = _reply.readString();
  ...
  return _result;
}
```

Proxy — GoF Object Structural

Consequences

– Additional overhead from indirection or inefficient proxy implementations



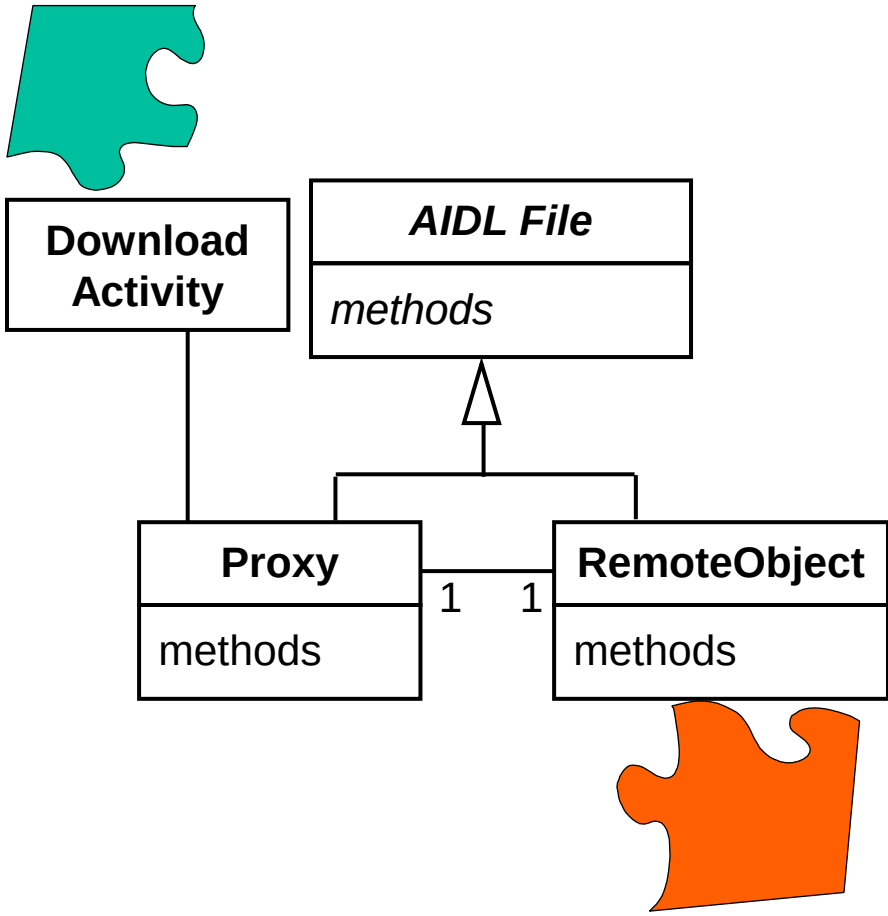

```
public String downloadImage(String uri) ... {
  android.os.Parcel _data =
    android.os.Parcel.obtain();
  android.os.Parcel _reply =
    android.os.Parcel.obtain();
  java.lang.String _result;
  _data.writeInterfaceToken(DESCRIPTOR);
  _data.writeString(uri);
  mRemote.transact(Stub.
   TRANSACTION_downloadImage, _data, _reply, 0);
  _reply.readException();
  _result = _reply.readString();
  ...
  return _result;
}
```

Proxy

GoF Object Structural

Consequences

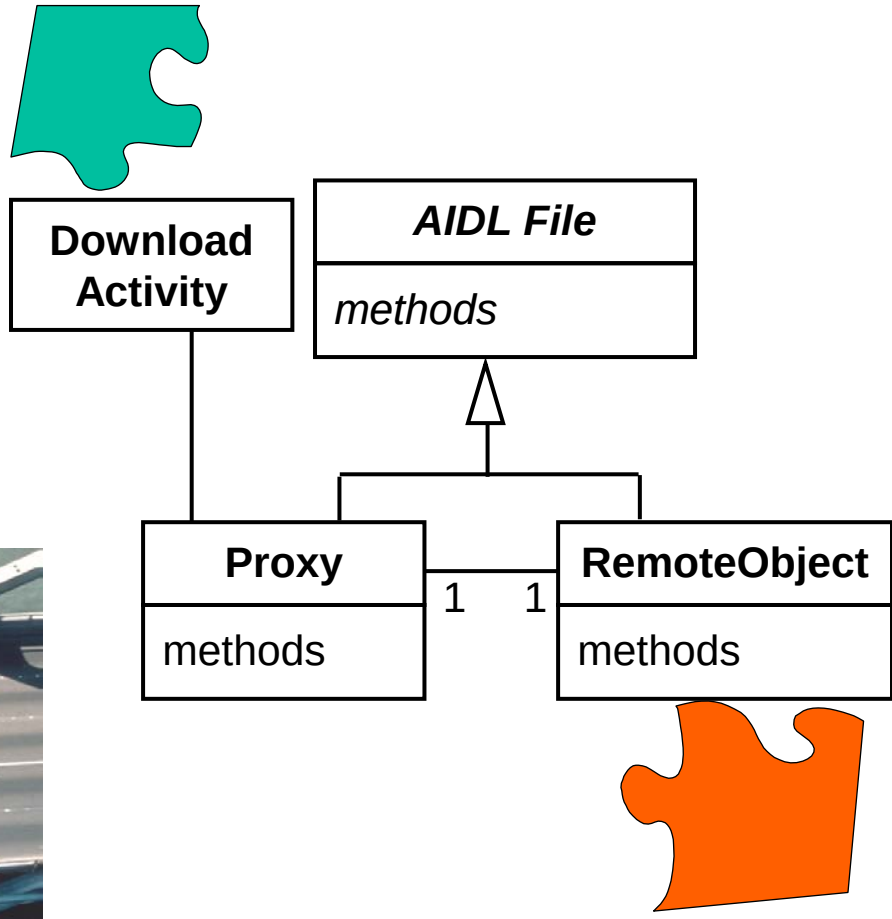
- Additional overhead from indirection or inefficient proxy implementations
- May impose overly restrictive type system



Proxy GoF Object Structural

Consequences

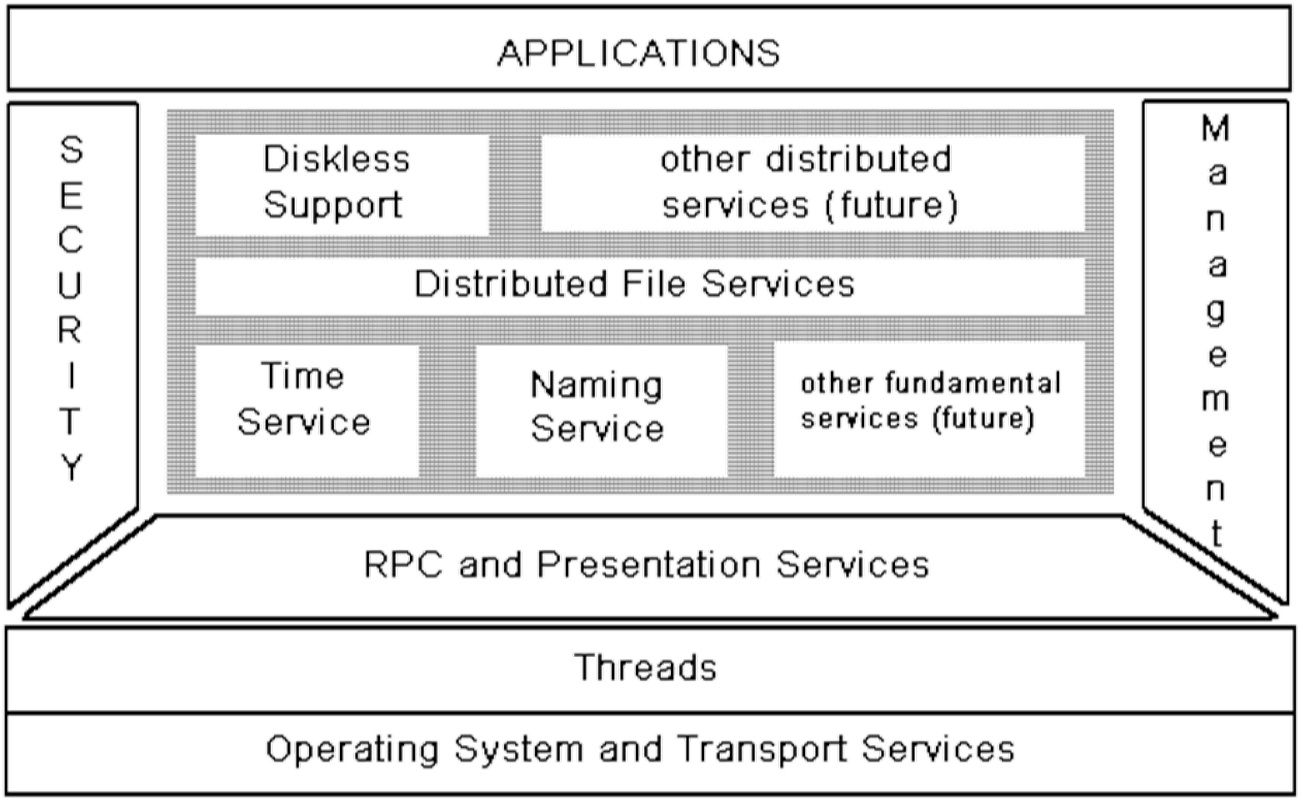
- Additional overhead from indirection or inefficient proxy implementations
- May impose overly restrictive type system
- It's not possible to entirely shield clients from problems with IPC across processes & networks



Proxy GoF Object Structural

Known Uses

- Remote Procedure Call (RPC) middleware
  - e.g., ONC RPC & OSF Distributed Computing Environment (DCE)

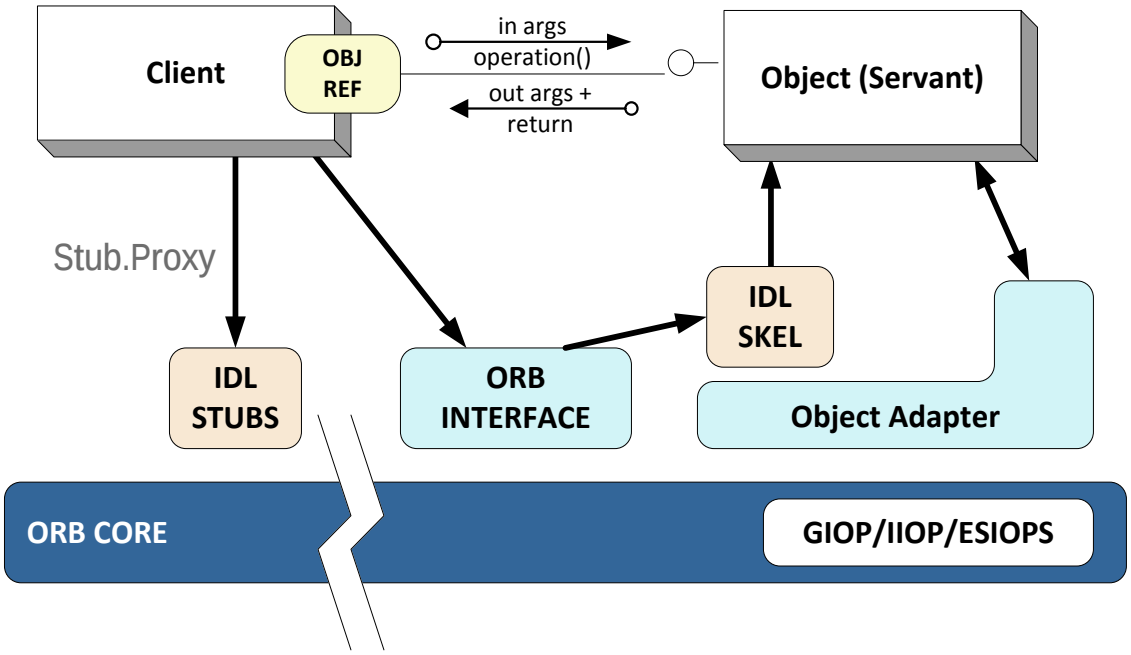


en.wikipedia.org/wiki/Distributed_Computing_Environment

Proxy GoF Object Structural

Known Uses

- Remote Procedure Call (RPC) middleware
- Distributed object computing middleware
  - e.g., Sun Java Remote Method Invocation (RMI) & OMG Common Object Request Broker Architecture (CORBA)

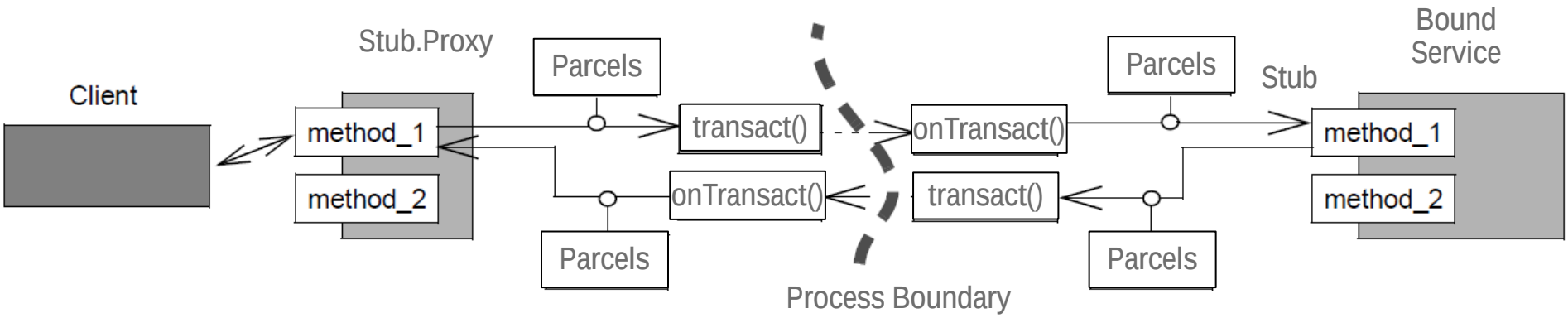


en.wikipedia.org/wiki/Object_request_broker

Proxy — GoF Object Structural

Known Uses

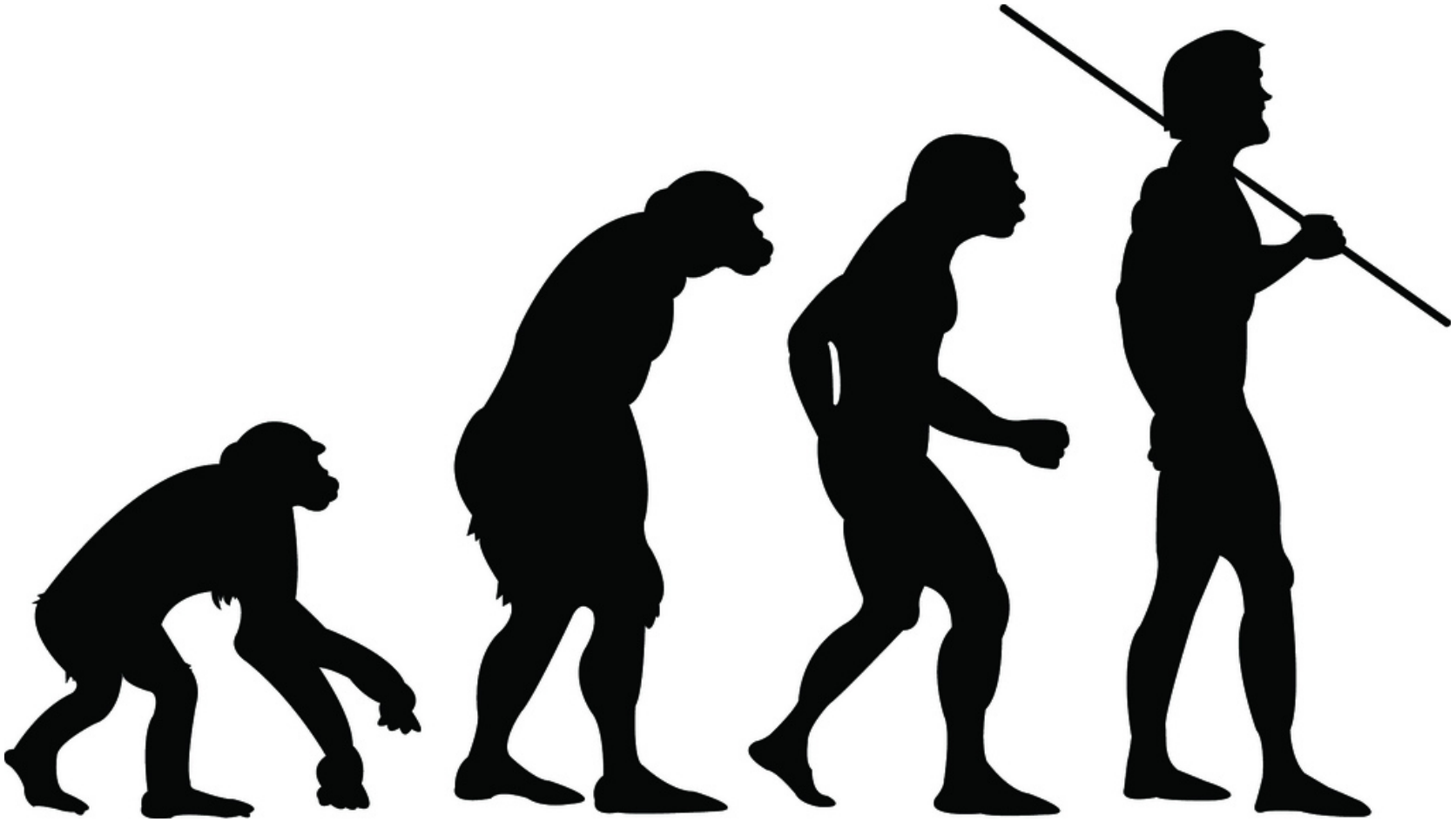
- Remote Procedure Call (RPC) middleware
- Distributed object computing middleware
- Local RPC middleware on smartphones
  - e.g., Android Binder



en.wikipedia.org/wiki/OpenBinder

Evolving Patterns into Programming Languages

- The *Iterator* pattern illustrates a recurring theme throughout the history of computing: *useful patterns evolve into programming language features*



Evolving Patterns into Programming Languages

- The *Iterator* pattern illustrates a recurring theme throughout the history of computing: *useful patterns evolve into programming language features*
- Assembly language patterns in the early days of computing led to language features in FORTRAN & C
  - e.g., closed subroutines & control constructs, such as loop, if/else, & switch statements

```
for (int i = 0; i < MAX_SIZE; ++i)
  ...

switch (tag_) {
case NUM: ...

if (6 == 9)
  printf ("I don't mind");
```

Evolving Patterns into Programming Languages

- The *Iterator* pattern illustrates a recurring theme throughout the history of computing: *useful patterns evolve into programming language features*
- Assembly language patterns in the early days of computing led to language features in FORTRAN & C
- Information hiding patterns done in assembly languages & C led to modularity features in Modula 2, Ada, C++, & Java
  - e.g., modules, packages, & access control sections

```
package com.example.expressiontree;

namespace std { …

class public class LeafNode extends ComponentNode {
   private int item;
   public int item()  { return item; }
   ...
```

Evolving Patterns into Programming Languages

- The *Iterator* pattern illustrates a recurring theme throughout the history of computing: *useful patterns evolve into programming language features*
- Assembly language patterns in the early days of computing led to language features in FORTRAN & C
- Information hiding patterns done in assembly languages & C led to modularity features in Modula 2, Ada, C++, & Java
- Preprocessor-style patterns in early C++ toolkits led to C++ templates & template meta-programming patterns are enhancing C++ templates

```
template <typename T>
class argv_iterator : public std::iterator
                                <std::forward_iterator_tag, T> {
public:
   argv_iterator (void) {}
   argv_iterator (int argc, char **argv, int increment);
   ...
```

Evolving Patterns into Programming Languages

- The *Iterator* pattern illustrates a recurring theme throughout the history of computing: *useful patterns evolve into programming language features*
- Assembly language patterns in the early days of computing led to language features in FORTRAN & C
- Information hiding patterns done in assembly languages & C led to modularity features in Modula 2, Ada, C++, & Java
- Preprocessor-style patterns in early C++ toolkits led to C++ templates & template meta-programming patterns are enhancing C++ templates
- Iterator patterns in C++ Standard Template Library (STL) led to built-in support for range-based for loops in C++11
  - e.g., languages like Java & C# also have built-in iterator support

```
for (auto &it : expr_tree)
  do_something (it);
```

C++11 range-based for loop

```
for(ExpressionTree it : exprTree)
  doSomething(it);
```

Java for-each loop

Not all patterns lend themselves to programming language support!