Android Concurrency: The Command Processor Pattern (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

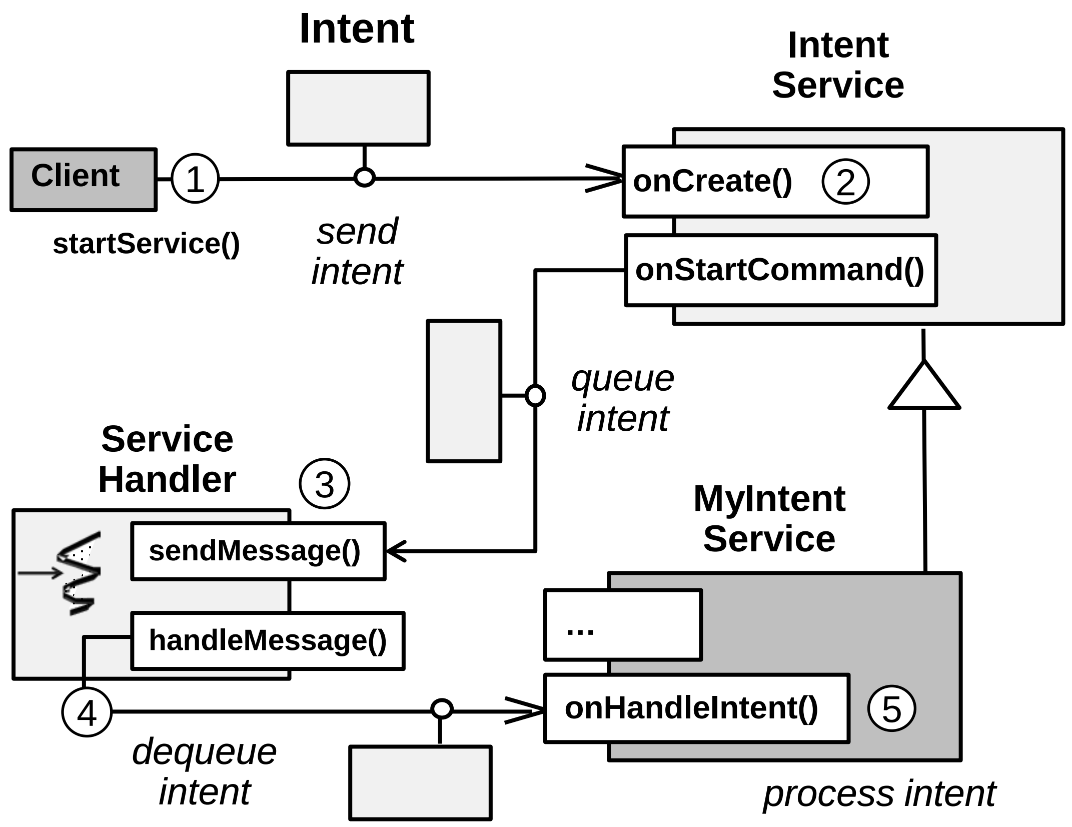
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand how *Command Processor* is applied in Android



Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
                post(Runnable r) {
    return sendMessageDelayed
      (getPostMessage(r), 0);
  }
```

frameworks/base/core/java/android/os/Handler.java has the source code

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
                post(Runnable r) {
    return sendMessageDelayed
      (getPostMessage(r), 0);
  }

  private final Message
    getPostMessage(Runnable r) {
      Message m =
        Message.obtain();
      m.callback = r;
      return m;
  }
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
    - It contains a Handler associated with the UI Thread's Looper
    - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
               post(Runnable r) {
    return sendMessageDelayed
      (getPostMessage(r), 0);
  }

  private final Message
    getPostMessage(Runnable r) {
      Message m =
        Message.obtain();
      m.callback = r;
      return m;
  }
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
           post(Runnable r) {
    return sendMessageDelayed
      (getPostMessage(r), 0);
  }
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
    sendMessageAtTime
      (Message msg,
       long uptimeMillis) {
    ...
    MessageQueue queue = mQueue;
    ...
    msg.target = this;
    queue.enqueueMessage
      (msg, uptimeMillis);
    ...
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
    sendMessageAtTime
      (Message msg,
       long uptimeMillis) {
    ...
    MessageQueue queue = mQueue;
    ...
    msg.target = this;
    queue.enqueueMessage
      (msg, uptimeMillis);
    ...
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public final boolean
    sendMessageAtTime
      (Message msg,
       long uptimeMillis) {
    ...
    MessageQueue queue = mQueue;
    ...
    msg.target = this;
    queue.enqueueMessage
      (msg, uptimeMillis);
    ...
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class MessageQueue {
  ...
  final boolean enqueueMessage
                   (Message msg,
                    long when) {
    final boolean needWake;
    synchronized (this) {
      Message p = mMessages;
      ...
      if (needWake) {
        nativeWake(mPtr);
    ...
```

Figure how where to enqueue the Message in the doubly-linked list

frameworks/base/core/java/android/os/MessageQueue.java has the source

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class MessageQueue {
  ...
  final boolean enqueueMessage
                      (Message msg,
                       long when) {
    final boolean needWake;
    synchronized (this) {
      Message p = mMessages;
      ...
      if (needWake) {
        nativeWake(mPtr);
    ...
```

frameworks/base/libs/utils/Looper.cpp has the source code

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue =
      me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
```

See previous part on Android Looper

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue =
      me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue =
      me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
```

frameworks/base/core/java/android/os/Looper.java has the source

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public void dispatchMessage
                 (Message msg) {
    if (msg.callback != null)
      handleCallback(msg);
    ...
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public void dispatchMessage
                (Message msg) {
    if (msg.callback != null)
      handleCallback(msg);
    ...
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
    - It contains a Handler associated with the UI Thread's Looper
    - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public void dispatchMessage
                (Message msg) {
    if (msg.callback != null)
      handleCallback(msg);
    ...
```

frameworks/base/core/java/android/os/Handler.java has the source

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public void dispatchMessage
                (Message msg) {
    if (msg.callback != null)
      handleCallback(msg);
    ...

  private final void
              handleCallback
               (Message message)
 {
   message.callback.run();
 }
```

Using Handler in Android to Post Runnables

- This example shows how to post Runnables to the UI Thread
- An Android Activity provides a single, focused thing a user can do
  - It contains a Handler associated with the UI Thread's Looper
  - runOnUiThread() uses this Handler to execute a specified action on the UI Thread

```
public class Handler {
  ...
  public void dispatchMessage
                (Message msg) {
    if (msg.callback != null)
      handleCallback(msg);
    ...

  private final void
                handleCallback
                 (Message message)
 {
   message.callback.run();
 }
```

This particular run() callback is executed in the UI Thread

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
  - Define an execute() operation if processing of a Command can be localized to one method

```
public class Intent implements
                 Parcelable, Cloneable {
   ...
}
```

frameworks/base/core/java/android/content/Intent.java has the source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
  - Can subclass the abstract Command class or some other means

```
public class Intent implements
         Parcelable, Cloneable {
  ...
  public Intent setData(Uri data)
  { /* ... */ }

  public Uri getData()
  { /* ... */ }

  public Intent putExtra
    (String name, Bundle value)
  {/* ... */ }

  public Object getExtra
    (String name)
  {/* ... */ }
}
```

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
  - Use patterns like Abstract Factory or Factory Method

```
public class AttachPhotoActivity
  extends ContactsActivity {
 ...
 private void  saveContact
   (ContactLoader.Result contact) {
   ...
   Intent intent =
     ContactSaveService.
       createSaveContactIntent
         (this, deltaList, "", 0,
           contact.isUserProfile(),
           null, null,
           raw.getRawContactId(),
           mTempPhotoFile.
             getAbsolutePath());
   startService(intent);
 }
 ...
```

Create the Intent Command

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
  - Use patterns like Abstract Factory or Factory Method
  - Determine a mechanism for passing the Command to the Executor

```
public class AttachPhotoActivity
  extends ContactsActivity {
  ...
  private void  saveContact
    (ContactLoader.Result contact) {
    ...
    Intent intent =
      ContactSaveService.
        createSaveContactIntent
          (this, deltaList, "", 0,
            contact.isUserProfile(),
            null, null,
            raw.getRawContactId(),
            mTempPhotoFile.
              getAbsolutePath());
    startService(intent);
  }
  ...
```

Pass the Intent Command to the designated Service via the Binder IPC mechanism

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
  - Provide the run-time environment for processing the Command object

```
public abstract class Context {
  public abstract void
   sendBroadcast(Intent intent);

  public abstract Intent
    registerReceiver
      (BroadcastReceiver receiver,
       IntentFilter filter);


  public abstract ContentResolver
    getContentResolver();

  ...

public abstract class Service
       extends ContextWrapper ...
{
  ...
}
```

frameworks/base/core/java/android/content/Context.java has the source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Receive the Command from the Creator

This hook method is called when a Service is created & spawns a background thread

```
public abstract class IntentService
                        extends Service {
  private volatile Looper
    mServiceLooper;
  private volatile ServiceHandler
    mServiceHandler;
  ...
  public void onCreate() {
    HandlerThread thr = new
      HandlerThread("IntentService["
                  + mName + "]");
    thr.start();

    mServiceLooper = thr.getLooper();
    mServiceHandler = new
      ServiceHandler(mServiceLooper);
  }
```

frameworks/base/core/java/android/app/IntentService.html has source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Enqueue the Command for subsequent processing

```
public abstract class IntentService
                         extends Service {
  private volatile Looper
    mServiceLooper;
  private volatile ServiceHandler
    mServiceHandler;
  ...
  public void onStart(Intent intent,
                            int startId) {
    Message msg = mServiceHandler.
                        obtainMessage();
    msg.arg1 = startId;
    msg.obj = intent;
    mServiceHandler.
      sendMessage(msg);
    }
  }
```

Enqueue the Intent Command in the ServiceHandler

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Dequeue the Command & initiate processing

```
public abstract class IntentService
                                extends Service {
  private volatile Looper
    mServiceLooper;
  private volatile ServiceHandler
    mServiceHandler;
  ...
  private final class ServiceHandler
                               extends Handler {
    public void handleMessage
      (Message msg) {
      onHandleIntent
        ((Intent)msg.obj);
      stopSelf(msg.arg1);
    }
  }
```

Dequeue the Intent Command & dispatch hook method

frameworks/base/core/java/android/app/IntentService.html has source code

Command Processor POSA1 Design Pattern

Implementation

- Define a class for Command execution that provides a generic interface used by the Executor
- Add state Concrete Commands need during their execution
- Define & implement the Creator
- Define the Execution Context
- Implement the Executor
  - Execute the Command in the Execution Context

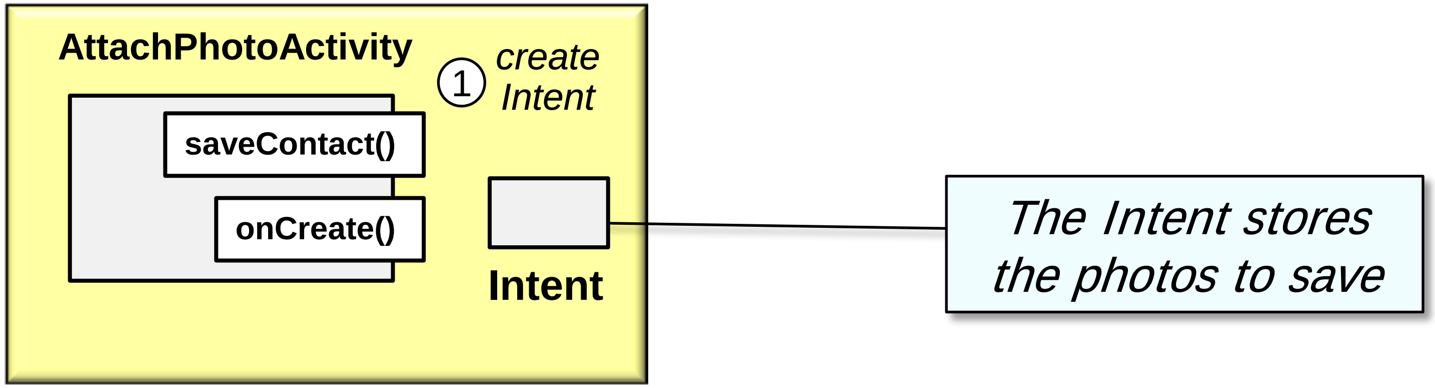
```
public class ContactSaveService
  extends IntentService {

  ...

  protected void
       onHandleIntent(Intent intent)
  {
    ... if (ACTION_SAVE_CONTACT.
              equals(action)) {
     saveContact(intent);
  }

  private void saveContact
                (Intent intent) {
    final ContentResolver resolver
      = getContentResolver();
  ...
}
```

Save the photo in the Contacts content provider

Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider

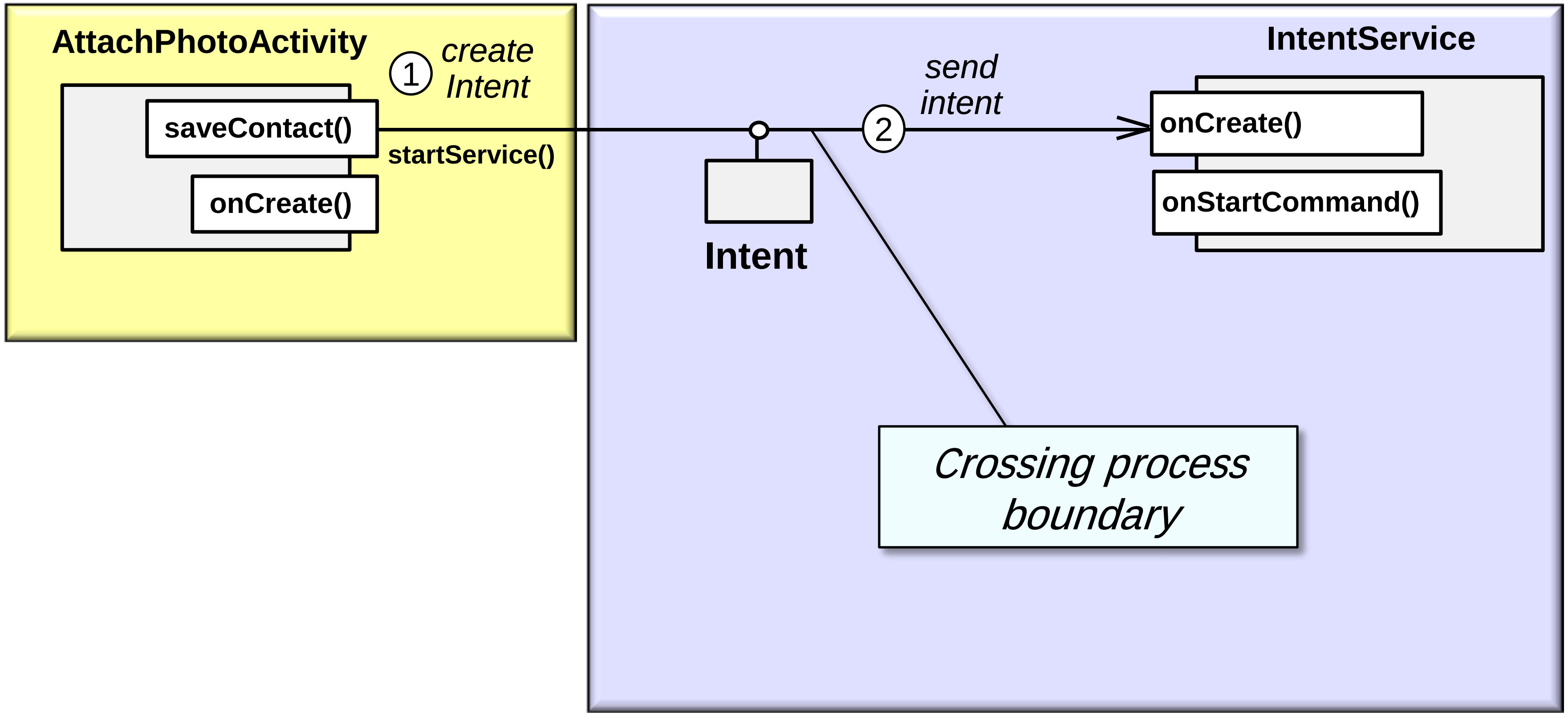


The Intent stores the photos to save

Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

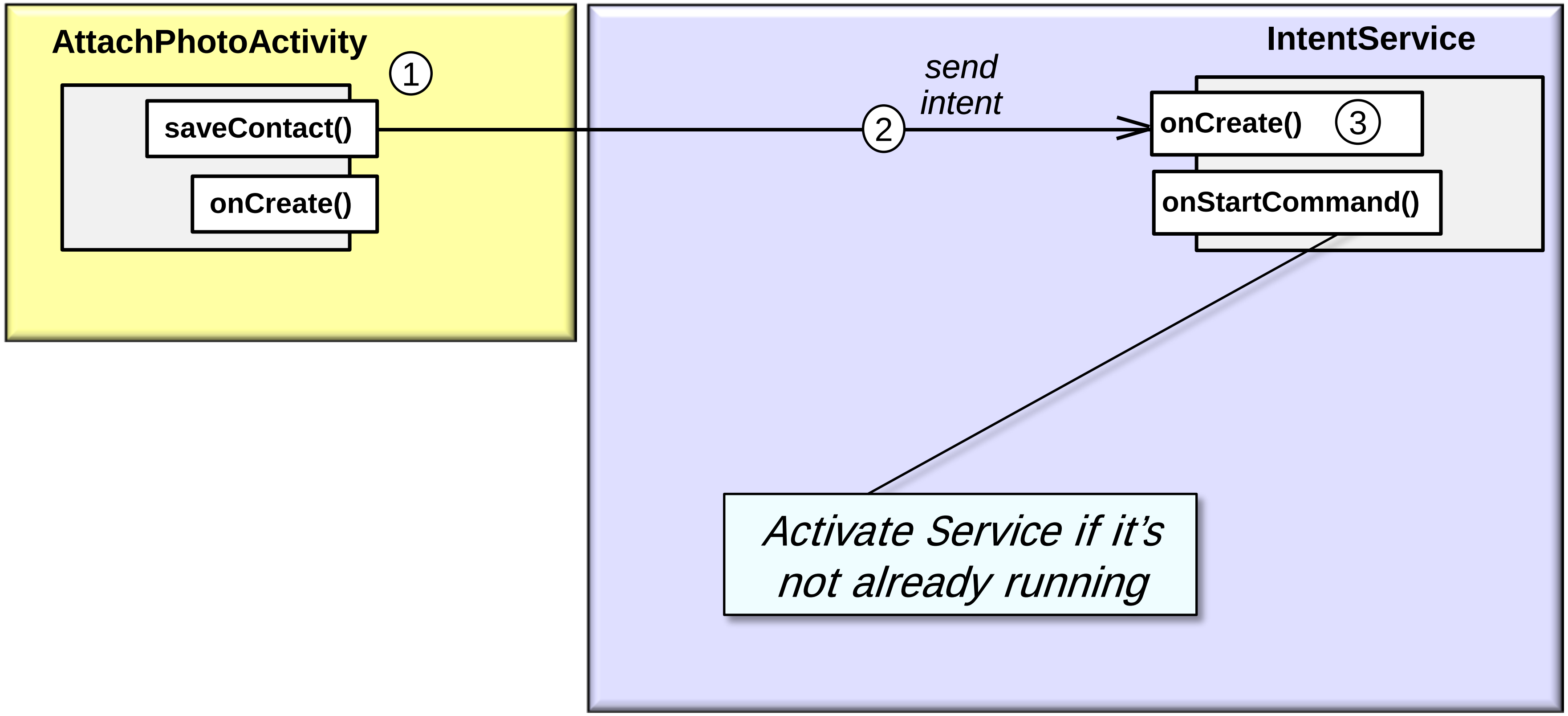
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

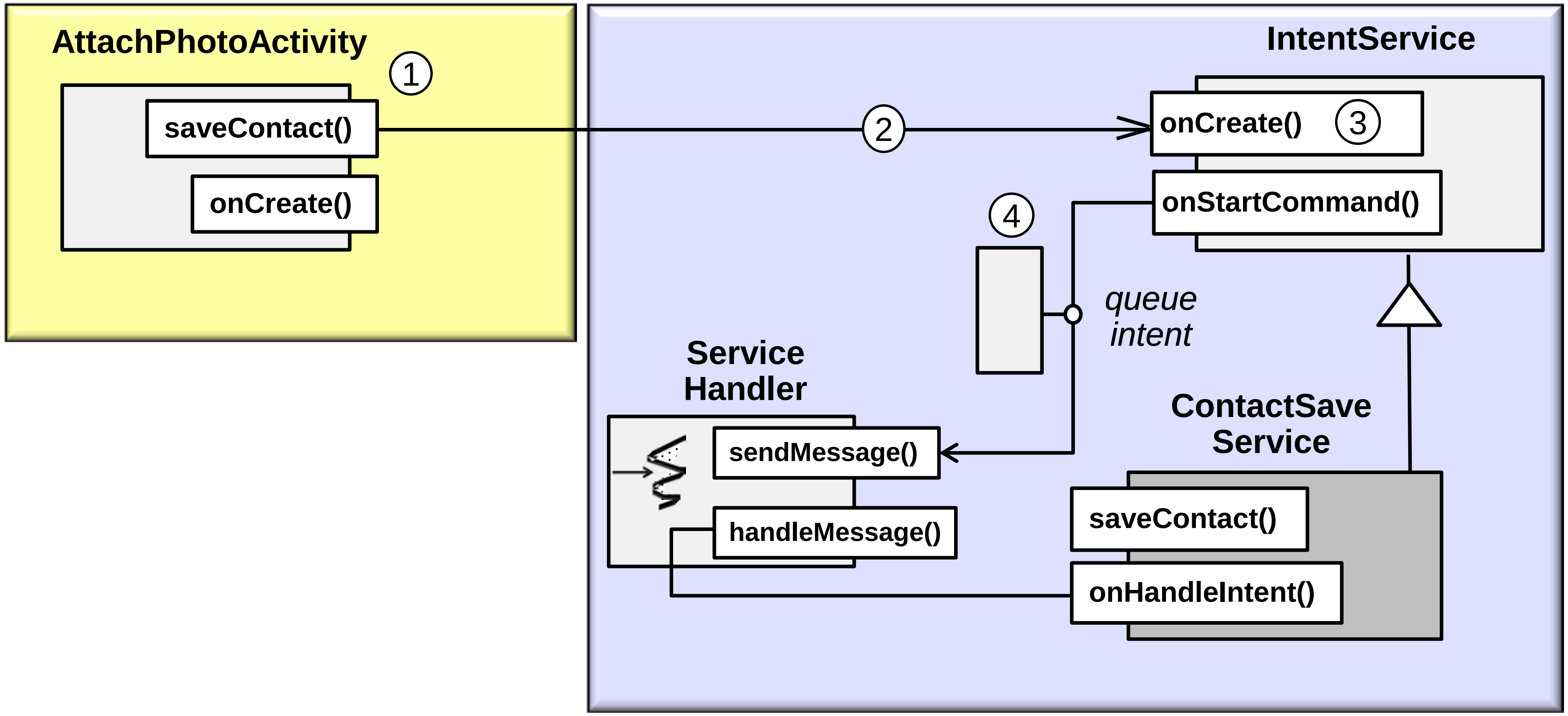
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

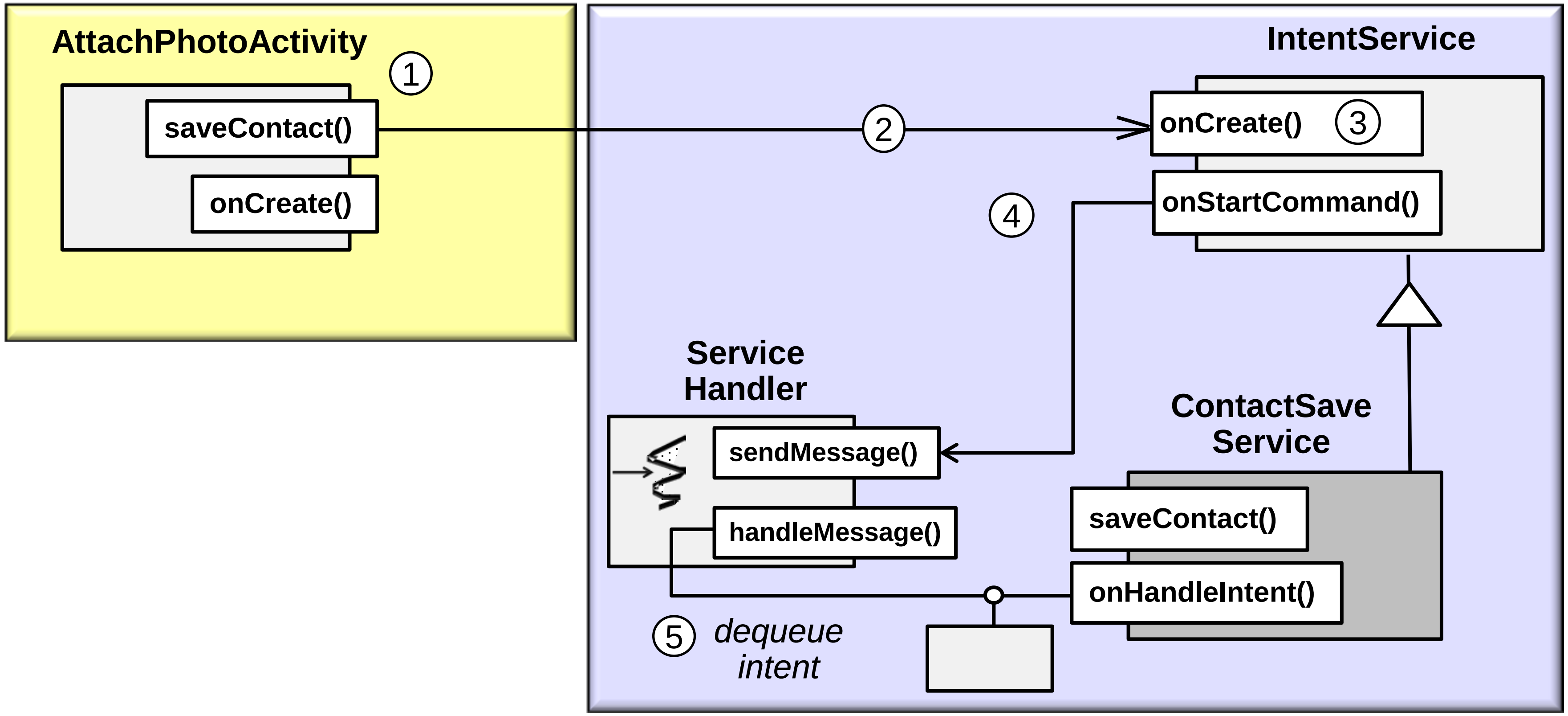
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Command Processor POSA1 Design Pattern

Applying the Command Processor pattern in Android

- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider

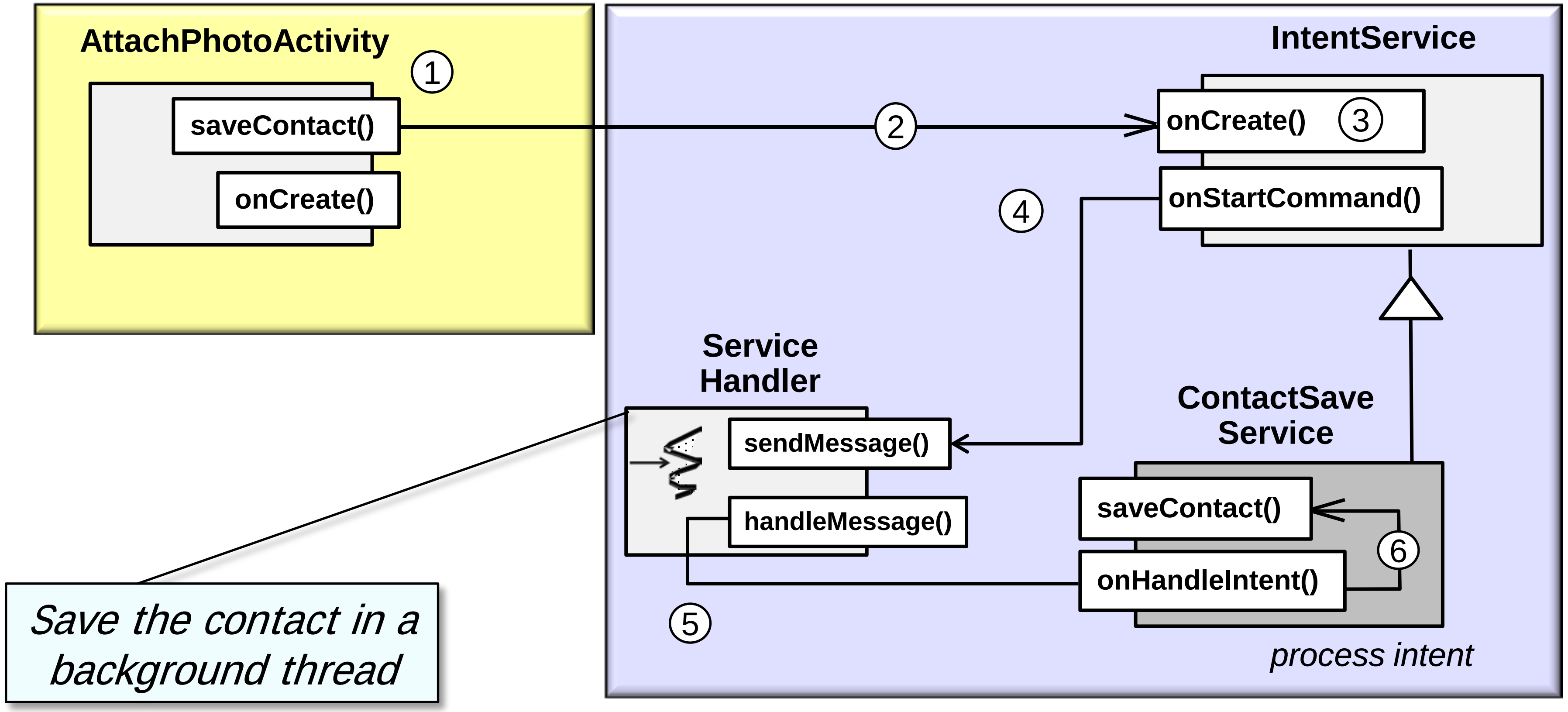


packages/apps/Contacts/src/com/android/contacts/activities/AttachPhotoActivity.java

Command Processor POSA1 Design Pattern

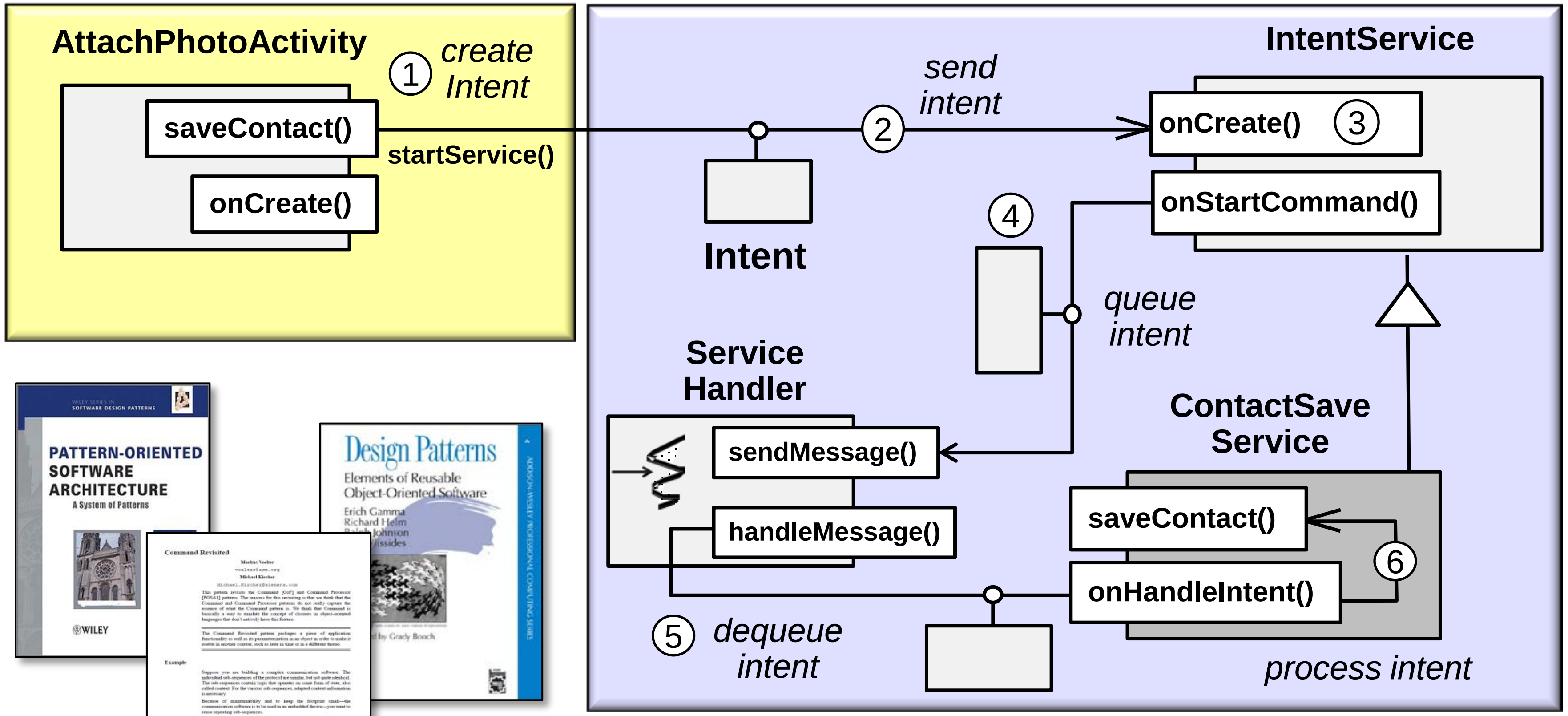
Applying the Command Processor pattern in Android

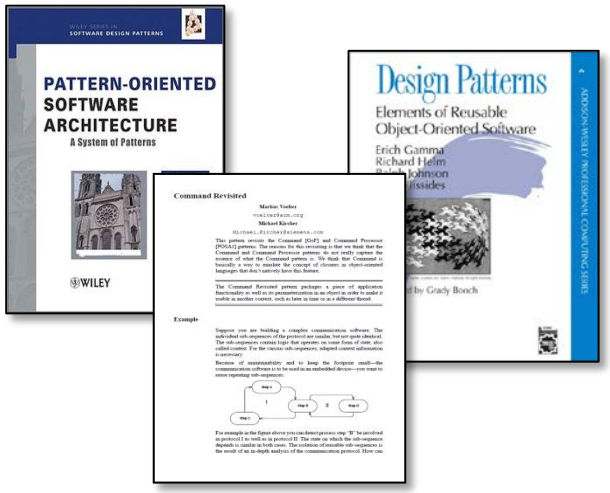
- AttachPhotoActivity allows Apps to attach images to contacts & then use the ContactSaveService to save changes to the Contacts content provider



Summary

- The Android Intent Service framework implements the *Command Processor* pattern & processes Intent Commands in a background Thread





Other patterns are involved here: *Activator*, *Messaging*, *Result Callback*, etc.