Android & Java Concurrency: the Active Object Pattern (Part 2)

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand how *Active Object* is implemented & applied in Android
Active Object

Implementation

- Implement the invocation infrastructure

```java
class Handler {
    boolean sendMessage (Message msg) {
        return sendMessageDelayed(msg, 0);
    }

    boolean sendMessageDelayed (Message msg, long delayMillis) {
        return sendMessageAtTime(msg, SystemClock.uptimeMillis() + delayMillis);
    }

    boolean sendMessageAtTime (Message msg, long uptimeMillis) {
        MessageQueue queue = mQueue;
        queue.enqueueMessage (msg, uptimeMillis);
        ...
    }
}
```

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

Implementation

- Implement the invocation infrastructure
- Implement the proxy
- Creates a concrete method request for each method invocation by a client

```java
class Handler {
    boolean sendMessage (Message msg) {
        return sendMessageDelayed(msg, 0);
    }

    boolean sendMessageDelayed (Message msg, long delayMillis){
        return sendMessageAtTime(msg,
            SystemClock.uptimeMillis() +
            delayMillis);
    }

    boolean sendMessageAtTime (Message msg, long uptimeMillis) {
        MessageQueue queue = mQueue;
        queue.enqueueMessage
            (msg, uptimeMillis);
        ...
    }
}
```

en.wikipedia.org/wiki/Proxy_pattern has more on the Proxy pattern
Active Object

Implementation

- Implement the invocation infrastructure
- Implement the proxy
- Implement the method requests
- Method requests can be considered as Messages

POS A2 Concurrency

```java
class Handler {
    boolean sendMessage (Message msg) {
        return sendMessageDelayed(msg, 0);
    }

    boolean sendMessageDelayed
        (Message msg, long delayMillis){
        return sendMessageAtTime(msg,
            SystemClock.uptimeMillis() +
            delayMillis);
    }

    boolean sendMessageAtTime
        (Message msg, long uptimeMillis) {
        MessageQueue queue = mQueue;
        queue.enqueueMessage
            (msg, uptimeMillis);
        ...
    }
```

Android Handler proxy & method requests are simpler than POSA active objects
Active Object                       POSA2 Concurrency

Implementation
• Implement the invocation infrastructure
• Implement the activation list

public class MessageQueue {
    ...
    final boolean enqueueMessage (Message msg, long when) {
        ...
    }

    final Message next() {
        ...
    }

    ...
}

frameworks/base/core/java/android/os/MessageQueue.java has source code
Active Object

Implementation

- Implement the invocation infrastructure
- Implement the activation list
  - Used to insert & remove a method request
  - Implemented as synchronized buffer shared between client threads & thread(s) where scheduler & servant runs

```java
public class MessageQueue {
    ...
    final boolean enqueueMessage (Message msg, long when) {
        ...
    }

    final Message next() {
        ...
    }
}
```
public class Looper {
    ...
    final MessageQueue mQueue;

    public static void loop() {
        ...
        for (;;) {
            Message msg = queue.next();
            ...
            msg.target.
            dispatchMessage(msg);
            ...
        }
        ...
    }

    ...

    frameworks/base/core/java/android/os/Looper.java has source code
Active Object

Implementation

- Implement the invocation infrastructure
- Implement the activation list
- Implement the scheduler
  - e.g., manages activation list & executes method requests that in the appropriate order

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

    public static void loop() {
        ...
        for (;;) {
            Message msg = queue.next();
            ...
            msg.target.
                dispatchMessage(msg);
        }
        ...
    }
    ...
}
```
Active Object

Implementation

- Implement the invocation infrastructure
- Implement the activation list
- Implement the scheduler
- Implement the servant

```java
class MessageHandler extends Handler {
    public void handleMessage (Message msg) {
        switch (msg.what) {
            case SET_PROGRESS_BAR_VISIBILITY:
                mAct.progress.setVisibility((Integer) msg.arg1);
                break;
            case PROGRESS_UPDATE:
                mAct.progress.setProgress((Integer) msg.arg1);
                break;
            ...
        }
    }
}
```
Active Object

Implementation

- Implement the invocation infrastructure
- Implement the activation list
- Implement the scheduler
- Implement the servant

A servant defines the behavior & state being modeled as an active object

class MessageHandler extends Handler {
    public void handleMessage (Message msg) {
        switch (msg.what) {
            case SET_PROGRESS_BAR_VISIBILITY:
                mAct.progress.setVisibility((Integer) msg.arg1);
                break;
            case PROGRESS_UPDATE:
                mAct.progress.setProgress((Integer) msg.arg1);
                break;
        }
    }
    ...
}
Active Object

Implementation

- Implement the invocation infrastructure
- Implement the activation list
- Implement the scheduler
- Implement the servant
- Determine rendezvous & return value policy

POSAS2 Concurrency

class MessageHandler extends Handler {
    public void handleMessage (Message msg) {
        ...
        Handler replyHandler = (Handler) msg.obj;

        Message reply = replyHandler.obtainMessage();
        reply.arg1 = // ...
        reply.arg2 = // ...

        reply.sendToTarget();
        ...
    }
}
## Implementation

- Implement the invocation infrastructure
- Implement the activation list
- Implement the scheduler
- Implement the servant
- Determine rendezvous & return value policy

The rendezvous policy determines how clients obtain return values from methods invoked on active objects.

```java
class MessageHandler extends Handler {
    public void handleMessage (Message msg) {
        ...
        Handler replyHandler = (Handler) msg.obj;
        Message reply = replyHandler.obtainMessage();
        reply.obj = // ...
        reply.arg1 = // ...
        reply.sendToTarget();
        ...
```
Implementation

- Implement the invocation infrastructure
- Implement the activation list
- Implement the scheduler
- Implement the servant
- Determine rendezvous & return value policy

  - The rendezvous policy determines how clients obtain return values from methods invoked on active objects

```java
class MessageHandler extends Handler {
    public void handleMessage (Message msg) {
        ...
        Handler replyHandler = (Handler) msg.obj;
        Message reply =
            replyHandler.obtainMessage();
        reply.obj = // ...
        reply.arg1 = // ...
        reply.sendToTarget();
        ...

    }
}
```

*A common idiom is to pass the original Handler via a Message to a background thread, which can then pass a response back to the original Handler*
Applying Active Object in Android
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread

See developer.android.com/reference/android/content/ContentResolver.html
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.
- ContentResolver provides apps access to an underlying ContentProvider.

Diagram:

- Activity
  - query()
- Content Resolver
  - Synchronous Query
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.
- ContentResolver provides apps access to an underlying ContentProvider.

Synchronous Query

Block Activity thread until the query is done

Run query on ContentProvider & return result
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.

See [developer.android.com/reference/android/content/AsyncQueryHandler.html](http://developer.android.com/reference/android/content/AsyncQueryHandler.html)
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.

Asynchronous Query

See developer.android.com/reference/android/content/AsyncQueryHandler.html
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.

Asynchronous Query

- Activity
  - startQuery()

- MyAsyncQueryHandler
  - sendMessage()

- Worker Handler
  - handleMessage()

- Content Resolver
  - query()

Run query on ContentProvider & return result

Block worker thread until the query is done

See developer.android.com/reference/android/content/AsyncQueryHandler.html
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.

See [developer.android.com/reference/android/content/AsyncQueryHandler.html](http://developer.android.com/reference/android/content/AsyncQueryHandler.html)
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread.

Asynchronous Query

Active Object                      POSA2 Concurrency

See developer.android.com/reference/android/content/AsyncQueryHandler.html
Applying Active Object in Android

- AsyncQueryHandler is a helper class that invokes ContentResolver calls asynchronously to avoid blocking the UI thread
- Internally, AsyncQueryHandler uses a (subset of the) Active Object pattern

```java
3. void loop() {
    ...
    for (;;) {
        Message msg = queue.next();
        ...
        msg.target.dispatchMessage(msg);
    }
    ...
}
```

```java
4. void handleMessage(Message msg) {
    switch (event) {
    case EVENT_ARG_QUERY:
        Cursor cursor = resolver.query(...);
        ...
    }
}
```

**Frameworks/base/core/java/android/content/AsyncQueryHandler.java** has code
Android’s HaMeR framework implements a subset of *Active Object* to enable communication between a client & a Handler running in separate threads.
Summary

- Android’s HaMeR framework implements a subset of Active Object to enable communication between a client & a Handler running in separate threads.
- It’s easy to extend Android with more sophisticated proxies & servants to create more complete Active Object implementations.

pragprog.com/magazines/2013-05/java-active-objects has more info
Summary

- Android’s HaMeR framework implements a subset of *Active Object* to enable communication between a client & a Handler running in separate threads.
- It’s easy to extend Android with more sophisticated proxies & servants to create more complete *Active Object* implementations.
- Java’s ExecutorService framework also implements the *Active Object* pattern.

End of Android & Java Concurrency: The Active Object Pattern (Part 2)