Android & Java Concurrency: the
Half-Sync/Half-Async Pattern (Part 1)

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 the *Half-Sync/Half-Async* pattern
Challenge: Combining Sync & Async Processing

Context

- A concurrent system that performs both asynchronous & synchronous processing services that must communicate
- The ThreadedDownloads app a good example of this context

See github.com/douglascraigschmidt/POSA-14/tree/master/ex/ThreadedDownloads
Challenge: Combining Sync & Async Processing

Problems

- Services that want the simplicity of synchronous processing shouldn’t need to address the complexities of asynchrony

```java
Bitmap downloadImage(String url) {
    InputStream is = (InputStream) new URL(url).getContent(); ...
    return BitmapFactory.decodeStream(is);
}
```

Each thread needs to block independently to prevent a flow-controlled connection from degrading the QoS that other clients receive.
Challenge: Combining Sync & Async Processing

Problems

- Services that want the simplicity of synchronous processing shouldn’t need to address the complexities of asynchrony.
- Synchronous & asynchronous processing services should be able to communicate without complicating their programming model or unduly degrading their performance.
Challenge: Combining Sync & Async Processing

Solution

- Decompose the services in the system into two layers: *synchronous* & *asynchronous*

A bounded # of threads in this layer can be mapped to separate CPUs/cores to scale up performance via concurrency
Challenge: Combining Sync & Async Processing

Solution

- Decompose the services in the system into two layers: *synchronous* & *asynchronous*
Solution

- Decompose the services in the system into two layers: synchronous & asynchronous
- Add a queueing layer between them to mediate the communication between services in the asynchronous & synchronous layers
Intent & Applicability of the Half-Sync/Half-Async Pattern
Half-Sync/Half-Async POSA2 Concurrency

**Intent**

- Decouple asynchronous (async) & synchronous (sync) service processing in concurrent systems by introducing two intercommunicating layers—one for async & one for sync service processing—to simplify programming without unduly reducing performance.

[Diagram of Half-Sync/Half-Async pattern with layers and services]

[www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf) has more info
Applicability

- When it’s necessary to make performance efficient & scalable, while also ensuring that the use of concurrency simplifies—rather than complicates—programming.
Applicability

- When it’s necessary to make performance efficient & scalable, while also ensuring that the use of concurrency simplifies—rather than complicates—programming
- When there are constraints on certain types of operations in certain contexts
  - e.g., short-duration vs. long-duration, non-blocking vs. non-blocking, etc.
Structure of the Half-Sync/Half-Async Pattern
Android & Java Concurrency: the Half-Sync/Half-Async Pattern (Part 1)

Half-Sync/Half-Async POSA2 Concurrency

Structure & Participants

- **Synchronous Service Layer**
  - Sync Service 1
  - Sync Service 2
  - Sync Service 3

- **Queueing Layer**
  - Queue
  - <<read/write>>

- **Asynchronous Service Layer**
  - Async Service
  - <<enqueue/dequeue>>
  - <<interrupt>>
  - External Event Source

- **UI Thread**
Android & Java Concurrency: the Half-Sync/Half-Async Pattern (Part 1)

Half-Sync/Half-Async  POSA2 Concurrency

Structure & Participants

Synchronous Service Layer
- Sync Service 1
- Sync Service 2
- Sync Service 3

Queueing Layer
- <<read/write>>
- <<dequeue/enqueue>>
- <<interrupt>>

Async Service

Asynchronous Service Layer

Queue

Blocking Queue

External Event Source
Half-Sync/Half-Async Structure & Participants

- **Synchronous Service Layer**:
  - Sync Service 1
  - Sync Service 2
  - Sync Service 3

- **Queueing Layer**:
  - Queue
  - <<read/write>>
  - <<read/write>>
  - <<read/write>>

- **Asynchronous Service Layer**:
  - Async Service
  - <<enqueue/dequeue>>
  - <<interrupt>>

- **External Event Source**

**Background threads**
Dynamics of the Half-Sync/Half-Async Pattern
Dynamics

<table>
<thead>
<tr>
<th>: External Event Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>notification</td>
</tr>
<tr>
<td>read()</td>
</tr>
<tr>
<td>message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Async Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>work()</td>
</tr>
<tr>
<td>message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue()</td>
</tr>
<tr>
<td>message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>: Sync Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>read()</td>
</tr>
<tr>
<td>notification</td>
</tr>
<tr>
<td>work()</td>
</tr>
</tbody>
</table>

An event occurs

If no input is available, the service thread blocks

Reactivates the service thread so that the synchronous read() continues
Event handling runs reactively/asynchronously
Half-Sync/Half-Async

**Dynamics**

- **External Event Source**
  - `read()`
  - `message` → `enqueue()`
  - `notification`

- **Async Service**
  - `work()`
  - `message`

- **Queue**
  - `enqueue()`
  - `message`
  - `notification`
  - `read()`
  - `message`
  - `enqueue()`
  - `work()`
  - `message`
  - `notification`

- **Sync Service**
  - `work()`
  - `message`

Queue requests without blocking caller

If no input is available, the service thread blocks

Reactivates the service thread so that the synchronous read() continues
Android & Java Concurrency: the Half-Sync/Half-Async Pattern (Part 1)

Dynamics

: External Event Source

: Async Service

: Queue

: Sync Service

Long-duration app processing runs synchronously

If no input is available, the service thread blocks
Reactivates the service thread so that the synchronous read() continues

Sync services run concurrently, relative both to each other & to async services
Consequences of the Half-Sync/Half-Async Pattern
Consequences
Consequences
+ Simplification & performance
  • Programming of higher-level sync processing services are simplified without degrading performance of lower-level system services
**Consequences**

+ Simplification & performance
+ Separation of concerns
  
  • Synchronization policies in each layer are decoupled so that each layer need not use the same concurrency strategies
Half-Sync/Half-Async

**Consequences**

+ Simplification & performance
+ Separation of concerns
+ Centralization of inter-layer communication

- Inter-layer communication is centralized because all interaction is mediated by the queueing layer
Consequences
Consequences

- May incur a boundary-crossing penalty
  - Arising from context switching, synchronization, & data copying overhead when data transferred between sync & async service layers via queueing layer

See [www.dre.vanderbilt.edu/~schmidt/PDF/INFOCOM-94.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/INFOCOM-94.pdf)
Consequences

- May incur a boundary-crossing penalty
- Higher-level app services may not benefit from async I/O

  • Depending on design of OS or app framework interfaces, higher-level services may not use low-level async I/O devices effectively
Half-Sync/Half-Async

Consequences
- May incur a boundary-crossing penalty
- Higher-level app services may not benefit from async I/O
- Complexity of debugging & testing
  - Apps can be hard to debug due to concurrent execution

Android & Java Concurrency: the Half-Sync/Half-Async Pattern (Part 1)
Known Uses of the Half-Sync/Half-Async Pattern
Known Uses

- UNIX Networking Subsystems

[Diagram showing synchronization and asynchronous processing]

www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf has more info
**Half-Sync/Half-Async**

**POSA2 Concurrency**

**Known Uses**
- UNIX Networking Subsystems
- Object Request Brokers (ORBs)

www.dre.vanderbilt.edu/~schmidt/PDF/OM-01.pdf has more info
Half-Sync/Half-Async

Known Uses
- UNIX Networking Subsystems
- Object Request Brokers (ORBs)
- Android AsyncTask framework
Summary
The Half-Sync/Half-Async pattern separates concerns between three layers, which makes concurrent software easier to understand, debug, & evolve.
Summary

- The Half-Sync/Half-Async pattern separates concerns between three layers, which makes concurrent software easier to understand, debug, & evolve.
- In addition, async & sync services don’t incur each other’s liabilities.
Summary

- The Half-Sync/Half-Async pattern separates concerns between three layers, which makes concurrent software easier to understand, debug, & evolve.
- In addition, async & sync services don’t incur each other’s liabilities, e.g.
  - Async service performance does not degrade due to blocking sync services.
The Half-Sync/Half-Async pattern separates concerns between three layers, which makes concurrent software easier to understand, debug, & evolve.

In addition, async & sync services don’t incur each other’s liabilities, e.g.

- Async service performance does not degrade due to blocking sync services
- The simplicity of programming sync services is unaffected by async complexities, such as explicit state management
Summary

- The Half-Sync/Half-Async pattern separates concerns between three layers, which makes concurrent software easier to understand, debug, & evolve.
- In addition, async & sync services don’t incur each other’s liabilities.
- The queueing layer avoids hard-coded dependencies between the async & sync service layers.
  - It’s also easy to reprioritize the order in which messages are processed.
End of Android & Java Concurrency: The Half-Sync/Half-Async Pattern (Part 1)