Overview of Layered Architectures

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 Lesson

1. Know what layered architectures are

- Application Layer
- Presentation Layer
- Session Layer
- Transport Layer
- Internet Layer
- Data Link Layer
- Physical Layer

- Applications
- Domain-Specific Middleware Services
- Common Middleware Services
- Distribution Middleware
- Host Infrastructure Middleware
- Operating Systems & Protocols
- Hardware
Learning Objectives in this Lesson

1. Know what layered architectures are

2. Understand the *Layers* architectural pattern
Learning Objectives in this Lesson

1. Know what layered architectures are
2. Understand the Layers architectural pattern
3. Recognize the layers in Android’s software stack
Learning Objectives in this Lesson

1. Know what layered architectures are
2. Understand the Layers architectural pattern
3. Recognize the layers in Android’s software stack
4. Realize why layering is used in Android
Overview of Layered Architectures
Layering is applied in many domains
Layering is applied in many domains, e.g.

- Computer networking protocol stacks

Enables end-to-end communication by specifying how data should be (un)packetized, addressed, transmitted, routed, & received

See [en.wikipedia.org/wiki/Protocol_stack](en.wikipedia.org/wiki/Protocol_stack)
Layering is applied in many domains, e.g.
- Computer networking protocol stacks
  - Lower layers handle interactions with the hardware
    - e.g., GSM, DSL, & Ethernet

See [en.wikipedia.org/wiki/Link_layer](en.wikipedia.org/wiki/Link_layer) & [en.wikipedia.org/wiki/Physical_layer](en.wikipedia.org/wiki/Physical_layer)
Layering is applied in many domains, e.g.

- Computer networking protocol stacks
  - Lower layers handle interactions with the hardware
  - Middle layers exchange packets across hosts & routers
    - e.g., IP, TCP, & UDP

See en.wikipedia.org/wiki/Internet_layer & en.wikipedia.org/wiki/Transport_layer
Layering is applied in many domains, e.g.

- Computer networking protocol stacks
  - Lower layers handle interactions with the hardware
  - Middle layers exchange packets across hosts & routers
  - Upper layers implement & interact with applications
    - e.g., PPTP, XDR, CDR, JSON

See [en.wikipedia.org/wiki/Session_layer](en.wikipedia.org/wiki/Session_layer) & [en.wikipedia.org/wiki/Presentation_layer](en.wikipedia.org/wiki/Presentation_layer)
Layering is applied in many domains, e.g.

- Computer networking protocol stacks
  - Lower layers handle interactions with the hardware
  - Middle layers exchange packets across hosts & routers
  - Upper layers implement & interact with applications
  - Applications (& middleware) mostly just deal with the upper layer(s)
    - e.g., FTP, TELNET, SMTP, & SNMP

See en.wikipedia.org/wiki/Application_layer
An Overview of Layered Architectures

- Layering is applied in many domains, e.g.
  - Computer networking protocol stacks
  - Communication middleware in multi-tier enterprise IT systems

Provides services beyond the operating system & protocol stacks to enable components in a distributed system to communicate & manage data

See en.wikipedia.org/wiki/Middleware_(distributed_applications)
Layering is applied in many domains, e.g.

- Computer networking protocol stacks
- Communication middleware in multi-tier enterprise IT systems
  - Lower layers provide portable APIs for accessing hardware & system resources
    - e.g., Linux, Windows, JVM, & ACE

See [www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf)
Layering is applied in many domains, e.g.
- Computer networking protocol stacks
- Communication middleware in multi-tier enterprise IT systems
  - Lower layers provide portable APIs for accessing hardware & system resources
  - Middle layers shield applications from network programming details
    - e.g., DDS, Web Services, MQTT, Spring, CORBA, etc.

See [www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf)
Layering is applied in many domains, e.g.
- Computer networking protocol stacks
- Communication middleware in multi-tier enterprise IT systems
  - Lower layers provide portable APIs for accessing hardware & system resources
  - Middle layers shield applications from network programming details
  - Upper layers enable domain-specific reuse of capabilities
    - e.g., MD-PnP, IIC, & FACE

See [www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf)
Layering is applied in many domains, e.g.

- Computer networking protocol stacks
- Communication middleware in multi-tier enterprise IT systems
  - Lower layers provide portable APIs for accessing hardware & system resources
  - Middle layers shield applications from network programming details
  - Upper layers enable domain-specific reuse of capabilities
- Applications may deal w/multiple layers

See www.dre.vanderbilt.edu/~schmidt/PDF/middleware-encyclopedia.pdf
Overview of the Layers Architectural Pattern
The concept of layering has been expressed as an architectural pattern.

See www.dre.vanderbilt.edu/~schmidt/POSA
The concept of layering has been expressed as an *architectural pattern*

"a structural organization schema for software systems that
• provides a set of predefined subsystems
• specifies their responsibilities &
• includes rules & guidelines for organizing the relationships between these roles’’

See [en.wikipedia.org/wiki/Architectural_pattern](http://en.wikipedia.org/wiki/Architectural_pattern)
The concept of layering has been expressed as an architectural pattern. An architectural pattern is a structural organization schema for software systems that:

- provides a set of predefined subsystems
- specifies their responsibilities &
- includes rules & guidelines for organizing the relationships between these roles

See en.wikipedia.org/wiki/Architectural_pattern
The concept of layering has been expressed as an *architectural pattern*

"a structural organization schema for software systems that
• provides a set of predefined subsystems
• specifies their responsibilities &
• includes rules & guidelines for organizing the relationships between these roles”

See [en.wikipedia.org/wiki/Architectural_pattern](en.wikipedia.org/wiki/Architectural_pattern)
The concept of layering has been expressed as an *architectural pattern*:

"a structural organization schema for software systems that
• provides a set of predefined subsystems
• specifies their responsibilities &
• includes rules & guidelines for organizing the relationships between these roles"

See [en.wikipedia.org/wiki/Architectural_pattern](en.wikipedia.org/wiki/Architectural_pattern)
The *Layers* architectural pattern has been described in various publications.

See [en.wikipedia.org/wiki/Multilayered_architecture](en.wikipedia.org/wiki/Multilayered_architecture)
An Overview of the Layers Architectural Pattern

- The *Layers* pattern structures software apps & infrastructure in several ways.

See posa1.blogspot.com/2008/05/layered-architecture-pattern.html
An Overview of the Layers Architectural Pattern

- The *Layers* pattern structures software apps & infrastructure in several ways:
  - a. Partitions an overall system architecture into groups of subtasks

![Diagram](image_url)
An Overview of the Layers Architectural Pattern

- The *Layers* pattern structures software apps & infrastructure in several ways
  a. Partitions an overall system architecture into groups of subtasks
  b. Decomposes groups of subtasks into levels of abstraction
An Overview of the Layers Architectural Pattern

- The *Layers* pattern helps to simplify software development & evolution
An Overview of the Layers Architectural Pattern

- The *Layers* pattern helps to simplify software development & evolution
- e.g., it replaces tightly coupled “big balls of mud”...

An Overview of the Layers Architectural Pattern

- The *Layers* pattern helps to simplify software development & evolution
- e.g., it replaces tightly coupled “big balls of mud”... with modular solutions that can be extended & contracted more easily

![Diagram of Layers Architectural Pattern](www.dre.vanderbilt.edu/~schmidt/family.pdf)
An Overview of the Layers Architectural Pattern

- Be careful when implementing a layered architecture to avoid unnecessary overhead when exchanging data between the layers.

See www.dre.vanderbilt.edu/~schmidt/PDF/p96-van_renessse.pdf
An Overview of the Layers Architectural Pattern

- Be careful when implementing a layered architecture to avoid unnecessary overhead when exchanging data between the layers
  - e.g., minimize context switching, synchronization, & data copying overhead

An Overview of Android’s Layered Architecture
Android’s architecture is structured in accordance to multiple layers.
An Overview of Android’s Layered Architecture

- Android’s architecture is structured in accordance to multiple layers

The Android Linux kernel controls hardware & manages system resources
An Overview of Android’s Layered Architecture

- Android’s architecture is structured in accordance to multiple layers

Several layers of middleware provide higher-level reusable services to apps
An Overview of Android’s Layered Architecture

- Android’s architecture is structured in accordance to multiple layers.

The application layer provides packaged functionality to end-users.
An Overview of Android’s Layered Architecture

- Layering is applied in complex systems like Android for several reasons
Layering is applied in complex systems like Android for several reasons, e.g.

- Enhance systematic software reuse

An intentional strategy for increasing productivity & improving software quality

See [en.wikipedia.org/wiki/Code_reuse#Systematic_software_reuse](en.wikipedia.org/wiki/Code_reuse#Systematic_software_reuse)
An Overview of Android’s Layered Architecture

- Layering is applied in complex systems like Android for several reasons, e.g.
  - Enhance systematic software reuse

\[\text{libC provides a common API for accessing OS kernel capabilities}\]
Layering is applied in complex systems like Android for several reasons, e.g.:

- Enhance systematic software reuse

An Overview of Android’s Layered Architecture

Enable apps to run concurrently over various types of multi-core hardware
Layering is applied in complex systems like Android for several reasons, e.g.:

- Enhance systematic software reuse
- Enable “plug & play” replacement of certain layer implementations
Layering is applied in complex systems like Android for several reasons, e.g.

- Enhance systematic software reuse
- Enable “plug & play” replacement of certain layer implementations

An Overview of Android’s Layered Architecture

- Applications
- Application Frameworks
- Native Libraries
- VM & Runtime
- Hardware Abstraction Layer
- Operating System Kernel
- Hardware

Shield apps from inconsistent hardware APIs
An Overview of Android’s Layered Architecture

- Layering is applied in complex systems like Android for several reasons, e.g.
  - Enhance systematic software reuse
  - Enable “plug & play” replacement of certain layer implementations

Effects of updates can be confined to the layer whose implementation changes
An Overview of Android’s Layered Architecture

- Layering is applied in complex systems like Android for several reasons, e.g.
  - Enhance systematic software reuse
  - Enable “plug & play” replacement of certain layer implementations
  - Reduce the complexity of APIs that app developers must understand

See [en.wikipedia.org/wiki/Facade_pattern](en.wikipedia.org/wiki/Facade_pattern)
Layering is applied in complex systems like Android for several reasons, e.g.

- Enhance systematic software reuse
- Enable “plug & play” replacement of certain layer implementations
- Reduce the complexity of APIs that app developers must understand
- Enable use of popular protocols, APIs, & programming languages
Layering is applied in complex systems like Android for several reasons, e.g.:

- Enhance systematic software reuse
- Enable “plug & play” replacement of certain layer implementations
- Reduce the complexity of APIs that app developers must understand
- Enable use of popular protocols, APIs, & programming languages
- These popular protocols & APIs are available in open-source form

See [source.android.com](source.android.com) & [source.android.com/source/building-kernels.html](source.android.com/source/building-kernels.html)
End of Overview of Layered Architectures