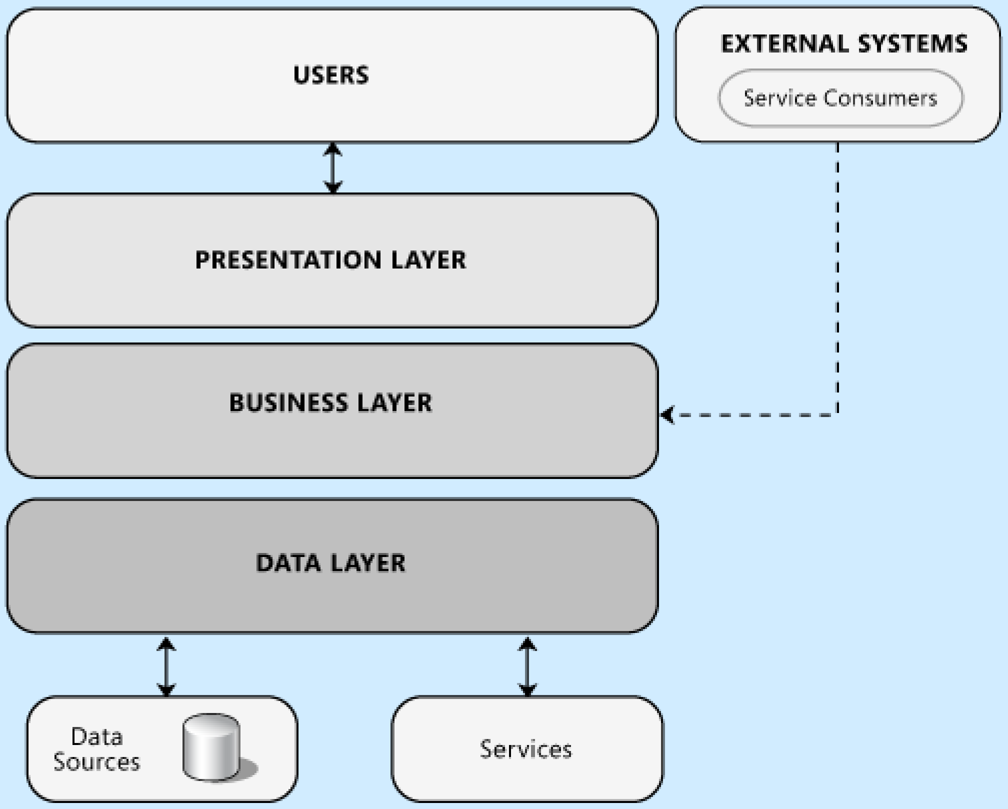
Sperasoft
Your game dev partner

N-Tier, Layered Design, SOA

Confusion

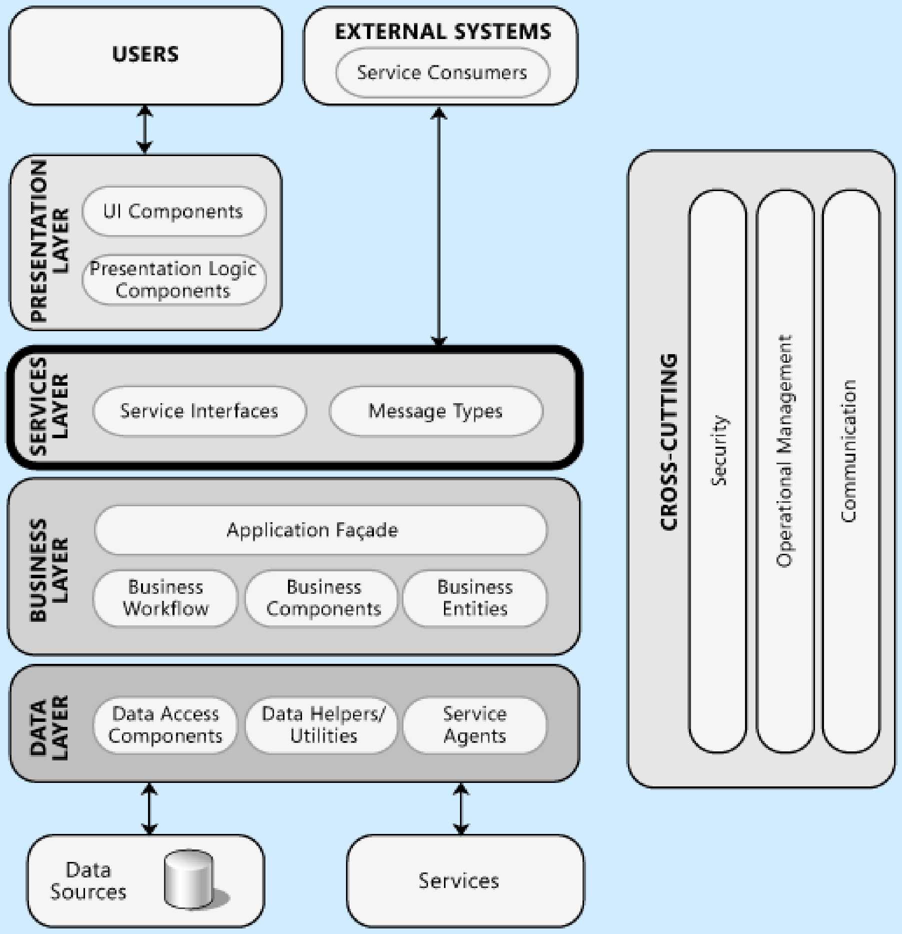
- **2-tier, 3-tier, N-tier**
- **Layered architecture** *(object-oriented design principle)*
- **Everywhere we hear of Presentation, Business & Data Access**

Presentation, Business & Data Layers



Reference: Layered Application Guidelines
http://msdn.microsoft.com/en-us/library/ee658109

When Service Layer Comes Up



Reference: Layered Application Guidelines
http://msdn.microsoft.com/en-us/library/ee658109

Distinction Between Layers & Tiers

- Both Layers and Tiers use the same set of names – Presentation, Business, Services, Data

- Layers describe the logical grouping of the functionality and components in an application

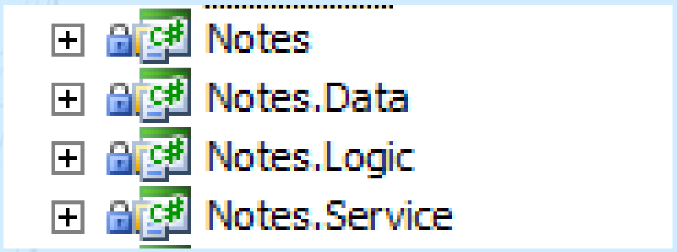
- **Tiers describe the physical distribution of the functionality and components on separate servers, computers, networks or remote locations**

- **Tiers imply a physical separation!**

- **Quite common to locate more then one layer on the same Tier (physical machine)**

Within a Single Component

Layered architecture becomes N-tier the following way:



Notes
Notes.Data
Notes.Logic
Notes.Service

Remember:
Tiers imply a physical separation!

Tiers

- Presentation (UI and API)
- Business Logic
- Data Access
- Data Storage (RDBMS, File system)

Presentation Tier

- Web UI
- Desktop applications
- iPhone, iPad, Android applications
- REST, SOAP interfaces etc
- Sockets, TCP, UDP etc
- FIX, Swift and other protocols

- Depends on platform and used technologies!

Presentation Tier (API)

Performs:

- **Depends on a host process**
- **Defines data tiers and other components to be used**
- **Builds up an application from components**
- **Defines used data sources**
  (connection strings, URLs, configuration)
- **Manage request context**
- **Stateless**

Presentation Tier (API)

- receives requests by a protocol
- converts requests into business objects
- invokes business logic
- gets the result from business logic
- prepares that result to be sent in a response
- catch exceptions from business logic
- defines how those exceptions are sent in response according to used protocol

Business Tier

Everything outside doesn’t matter:

- *Host process agnostic*
- *Data access agnostic*

It means:

- *Shouldn’t know about Presentation & Data Tiers*

Business Tier (cont'd)

- Defines Business logic Public interface
- Defines Data access Public interface
- Implements Business logic
- Defines Business objects (business entities)
- Defines meaningful Exceptions (business related)

Business Tier (cont'd)

Performs:

- Business processing, business workflow
- Authentication and authorization
- Auditing
- Exception handling (log and re-throw)
- Maintain Transactionality (open, commit, rollback)

Business Entities

- Are defined in Business Tier
- NEVER exposed via the Presentation Tier
- NEVER assume data storage mechanism

It means:

- No annotations to define XML, JSON, SOAP, SQL Tables, SQL parameters and so one

Data Access Tier

- Implements Data Access Interface defined in Business Tier
- Tightly coupled with Data Storage Tier
- Call chain neutral
- Stateless
- Transactionalable

Data Storage Tier

- RDBMS, Object-Oriented DBMS, NoSQL DB
- File system, NAS, DAS
- Remote storages: Clouds, FTP etc
- In-Memory!

No business logic here!

All Things Together

- Single implementation of Business Tier
- Multiple implementations of Presentation Tier
- Multiple implementations of Data access tier
- Multiple implementations of Data Storage Tier

Benefits

- We can mix components (tiers) to deploy required service (select SQL storage and REST API)
- Implementation process is Business logic centric
- Scalability up and out
- Extensibility
- Maintainability
- Reusability
- Testability (unit and integration tests)
- Data Tier performance (isolated to optimizations)

Drawbacks

- **Traceability - use EAI patterns and tools)**
- **Complexity - simple operation requires lots of code for all tiers)**
- **Many similar classes - each class contains the tier specific representation of data)**
- **Performance (under question)**

Patterns

Request-Reply:

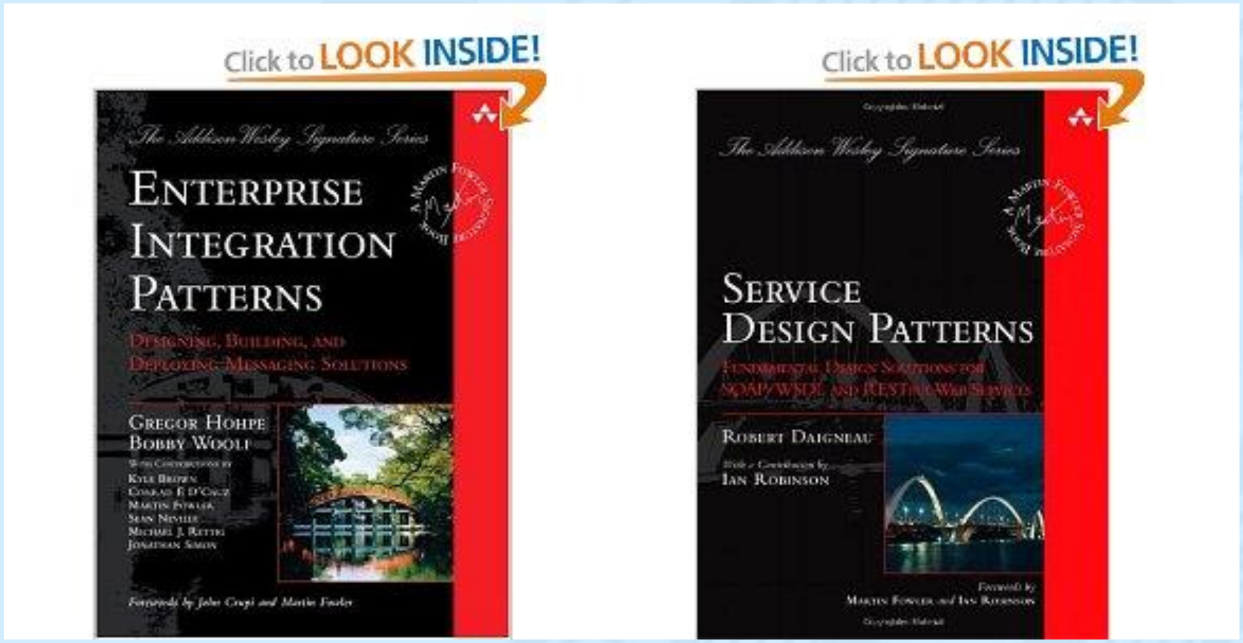
- **When an application sends a message, how can it get a response from the receiver?**
- **Send a pair of *Request-Reply* messages, each on its own channel.**

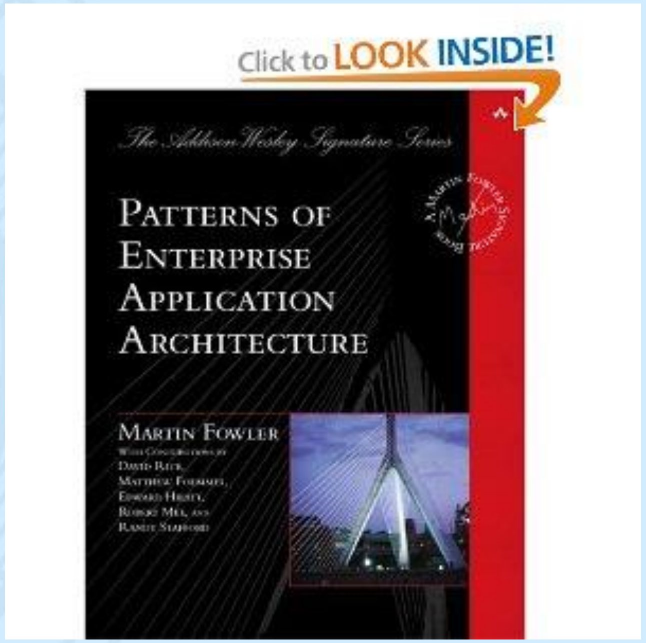
Patterns (cont'd)

Correlation Identifier:

- **How does a requestor that has received a reply know which request this is the reply for?**
- **Each reply message should contain a *Correlation Identifier*, a unique identifier that indicates which request message this reply is for.**

Books


Click to LOOK INSIDE!
The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
Click to LOOK INSIDE!
The Addison-Wesley Signature Series
Service Design Patterns
Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services
Robert Daigneau
Ian Robinson
Foreword by Martin Fowler and Ian Robinson


Click to LOOK INSIDE!
The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
Martin Fowler

SOA

- SOA comes up to divide Middleware to separate services
- N-tier implementation is in Services grouped by Purpose

Good Practice

- Think of your application as of an application that have both: UI of any kind and public API
- UI should NOT have any logic
- UI should ONLY use public API for everything
- That API becomes Middleware services and encapsulates all logic
- Extract “elemental” service out of it and build the Data access Tier of services