Building End-to-End Multi-Client Service Oriented Applications

Module 02

SOA & Technology

pluralsight
hardcore developer training

Service Oriented Architectures & Applications

- **"The decomposition of a system into autonomous or nearly autonomous units of responsibility and exposure."**
  - Decomposition can take various forms (functional, volatility-based)
  - Either case, services "orchestrate/manage" and "expose" functionality
- **SO-Applications are based on loosely-coupled services**
  - Standards & protocols couple services together
  - Technologies sit behind them
- **A SO-Application is essentially an API exposing functionality to the outside world through a set of standards and protocols**
- **SOA evolved from previous programming paradigms**

Evolution of Software Development

Procedural Programming

Applications were developed by continuously calling functions based on steps to be taken in order to complete a given task or requirement.

Problems:

- No reuse outside of application.
- No design analogy to real-world made for cumbersome development process.

Evolution of Software Development

Object Oriented Programming

Applications were built by working with various entities that resembled real-world counterparts, housing both data and behavior.

Problems:
- No reuse outside of application.
- Required lots of plumbing for managing ancillary functionality *(reliability, transactions, security, etc.)*.

Procedural Programming

Applications were developed by continuously calling functions based on steps to be taken in order to complete a given task or requirement.

Problems:
- No reuse outside of application.
- No design analogy to real-world made for cumbersome development process.

Evolution of Software Development

Component Oriented Programming

Objects could now be encapsulated and managed by a common abstraction layer providing loose-coupleness (COM) and housed in separate libraries (DLLs).

Problems:
- Ancillary functionality typically required external service management *(MTS, COM+, Corba)*.

Procedural Programming

Applications were developed by continuously calling functions based on steps to be taken in order to complete a given task or requirement.

Problems:
- No reuse outside of application.
- No design analogy to real-world made for cumbersome development process.

Object Oriented Programming

Applications were built by working with various entities that resembled real-world counterparts, housing both data and behavior.

Problems:
- No reuse outside of application.
- Required lots of plumbing for managing ancillary functionality *(reliability, transactions, security, etc.)*.

Evolution of Software Development

Service Oriented Programming

Objects & Components are managed and encapsulated behind autonomous services, each with its own set of responsibilities.

Problems:

- Needs a good "glue" technology to handle cumbersome plumbing for ancillary technologies.
- Need to accommodate interoperability.

Enter WCF & Web API !

Procedural Programming

Applications were developed by continuously calling functions based on steps to be taken in order to complete a given task or requirement.

Problems:
- No reuse outside of application.
- No design analogy to real-world made for cumbersome development process.

Object Oriented Programming

Applications were built by working with various entities that resembled real-world counterparts, housing both data and behavior.

Problems:
- No reuse outside of application.
- Required lots of plumbing for managing ancillary functionality *(reliability, transactions, security, etc.)*.

Component Oriented Programming

Objects could now be encapsulated and managed by a common abstraction layer (COM) and housed in separate libraries (DLLs).

Problems:
- Ancillary functionality typically required external service management *(MTS, COM+, Corba)*.

Characteristics of Services

- **Simple:**
  - Retrieve product record for given ID
    - In this case, service simply exposes API for simple data access
- **Complex:**
  - Process Order
    - Receive packaged information for order (cart info, customer, billing)
    - Update data records for product inventory and customer history
    - Process customer credit card information
    - Build invoice and email to customer

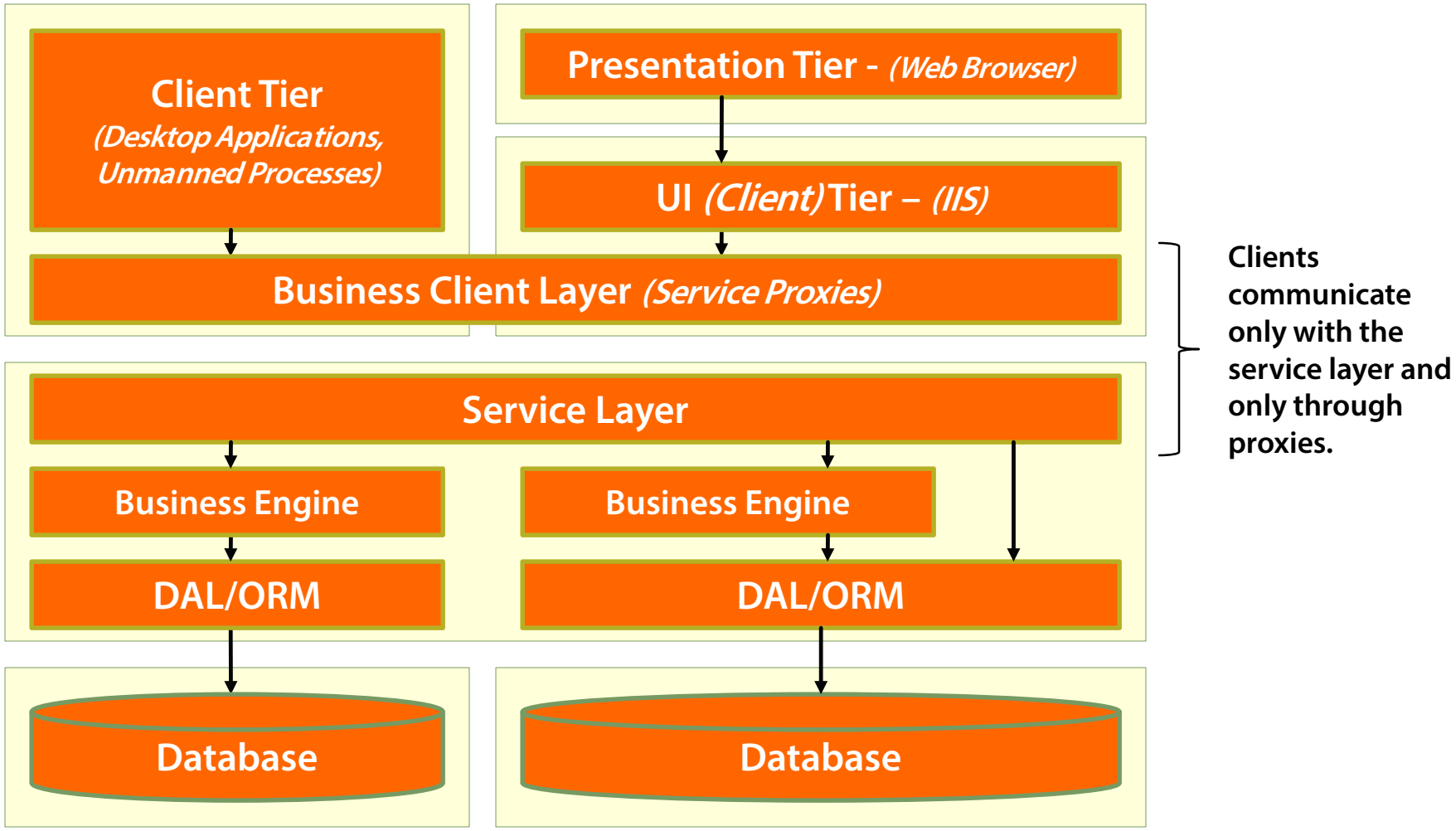
- **Either case should be simple to call from client**
  - Provide the API of the back-end to the client
  - Single point of entry for client

Characteristics of Services

- **Have all the characteristics of a full-blown application**
  - Should be fault-handled
  - Should always leave system in consistent state (transactional)
  - Should be secure
  - Should handle threading properly

- **A black-box unit of work access-point**

A Typical Architecture



Service Technologies

- **WCF**
  - Still the preferred technology for writing rich services
  - Provides robust model for exposing services and rich option-set
  - Still has interoperability limitations to poorly-tooled clients
- **Web API**
  - Less rich in offered service characteristics
  - Promotes a REST-oriented design
  - Offers great interoperability

- **Most good systems will combine both**

Other Technologies I will be using

- **SQL Server**
- **Entity Framework (code-first)**
- **Managed Extensibility Framework**
- **ASP.NET MVC**
- **Knockout JS**
- **Bootstrap**
- **Windows Presentation Foundation**
- **Assorted libraries throughout**
  - Moment (date handling in JS)
  - MahApps (WPF styling)
  - ToastR (JS notification display)
  - Fluent Validation (rules engine)

End of module

ALL THIS AND AN APP TOO !