Patterns of Implementation

Miguel A. Castro
@miguelcastro67

The Most Popular Question

- Now What?
- Knowing a technology vs. implementing in the real world
- Gotchas to know about
- Techniques & practices to make life easier

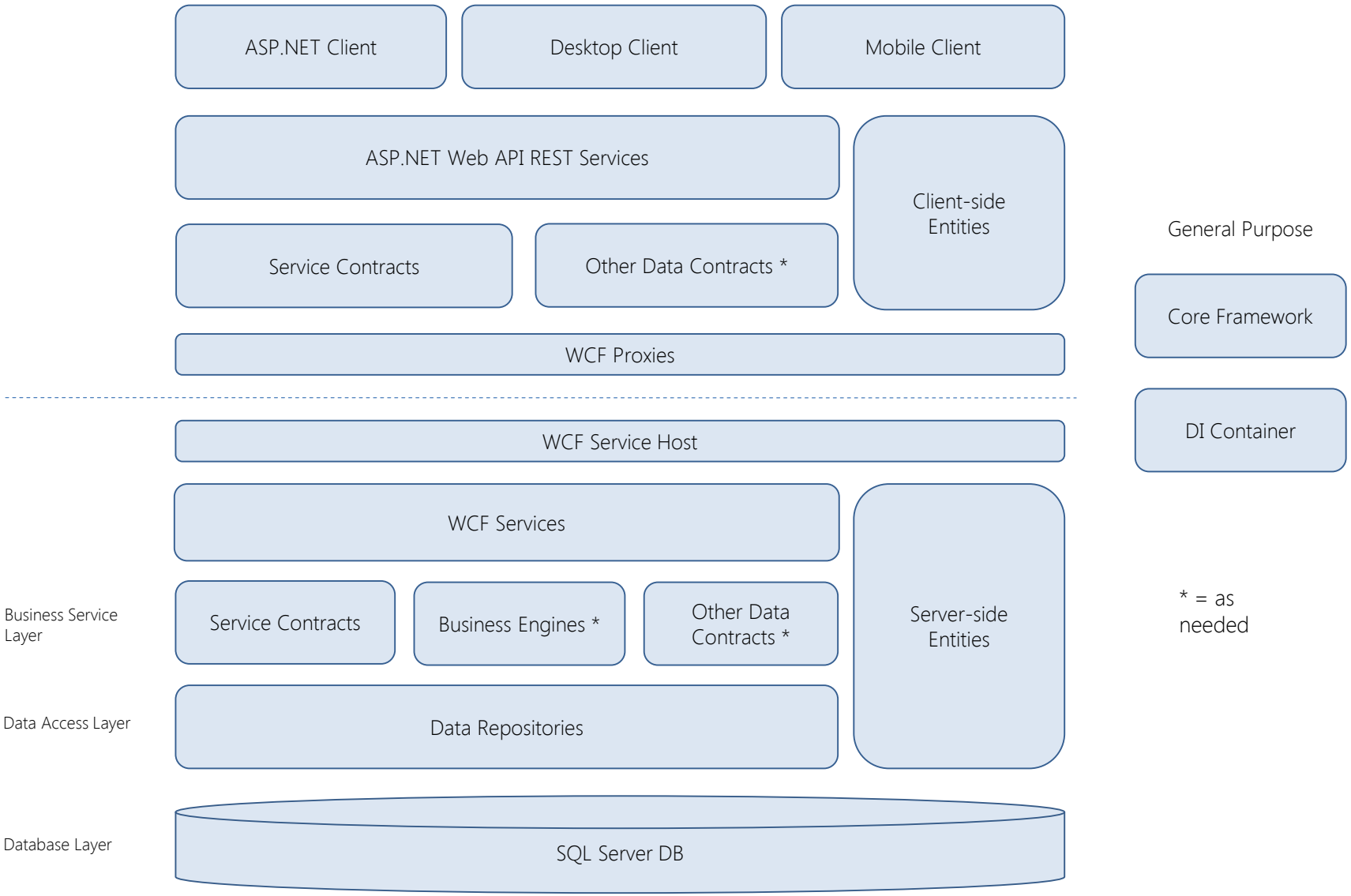
Layers

- **Less strict about one-step layer-to-layer**
  - Still strict about one-direction layer reference
- **Many times a layer just pass-through**
- **Business engines introduced as needed**
- **Perfectly acceptable for services to speak directly to data layer**
- **No need to repeat objects in each layer**
  - Data entities -> Business objects -> DataContracts
  - Except to explicitly create data contracts for good reason

Entities & Data Contracts

- **Can travel up and down each side of wire**
- **Two sets: business & client**
- **Business**
  - Simple and lean
  - EF Code-First
  - Used from data layer to service data contracts
  - Data DB-Context sets up ORM rules among entities
- **Client**
  - Richer and bindable (property notification)
  - Can contain validation (data annotations)
- **Data Contracts used when passing entity across not feasible**
  - Too much or too little info
  - Need combination of entities
  - Flatter structure (entities may support relationships)

Architecture



Demo Time

Service Contracts, Services, & DI

- **Service contracts use each side's entities**
  - Will have two sets of contracts
  - Can set namespace using **ContractNamespace** attribute
- **Need to write WCF Instance Provider to get service from container**
  - Most containers have NuGet package for WCF integration
- **All down-level layers injected**
  - Data Repositories
  - Business Engines
- **Can use mocking for unit testing**

Demo Time

Client Access Patterns

- **MVC & API controllers can use DI for proxies**
  - Inject service contracts
- **XAML ViewModels can use DI**
  - Inject proxies
  - Use service locator for on-demand proxies

To Be Continued…

- **All this and more is built piece by piece**
  - Building Multi-Client End-to-End Service Oriented Applications