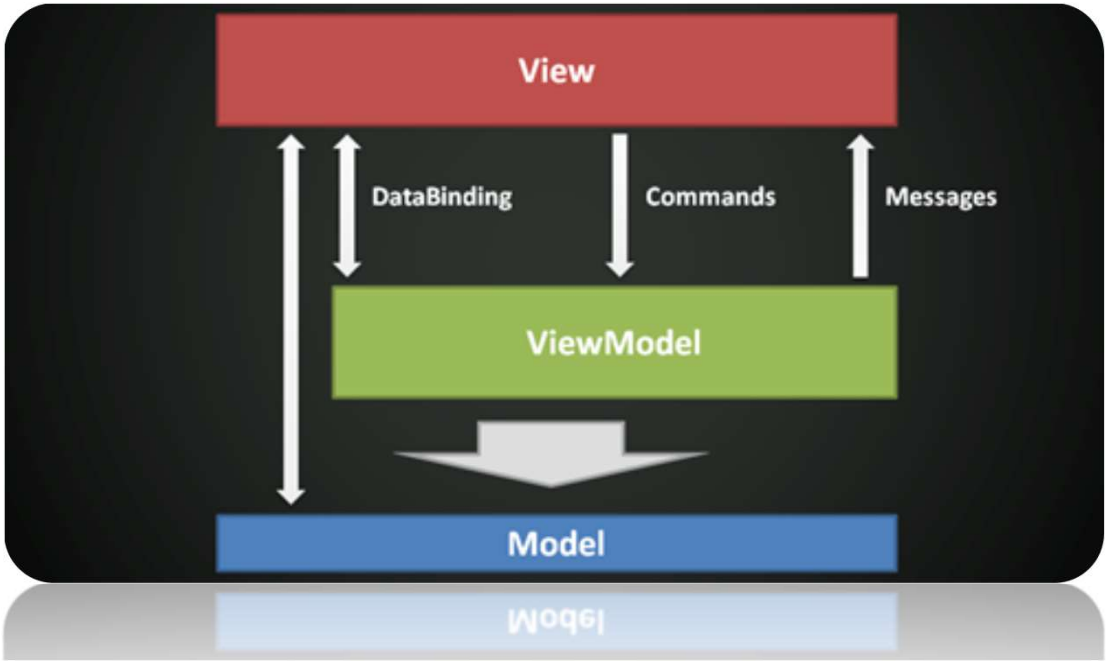
Model-View-ViewModel

View: XAML

ViewModel e Model: C#



icubed

MVVM

Obiettivi:

- Facilitare il paradigma designer/developer
- Stratificazione e ordine
- Testing del codice

Mezzi:

- Tutti già inclusi in WPF
- Niente framework
  - Alcuni aiuti come MVVM Light Toolkit, Prism
- Dependency Injection e AOP con Unity, MEF, Castle, Spring.NET

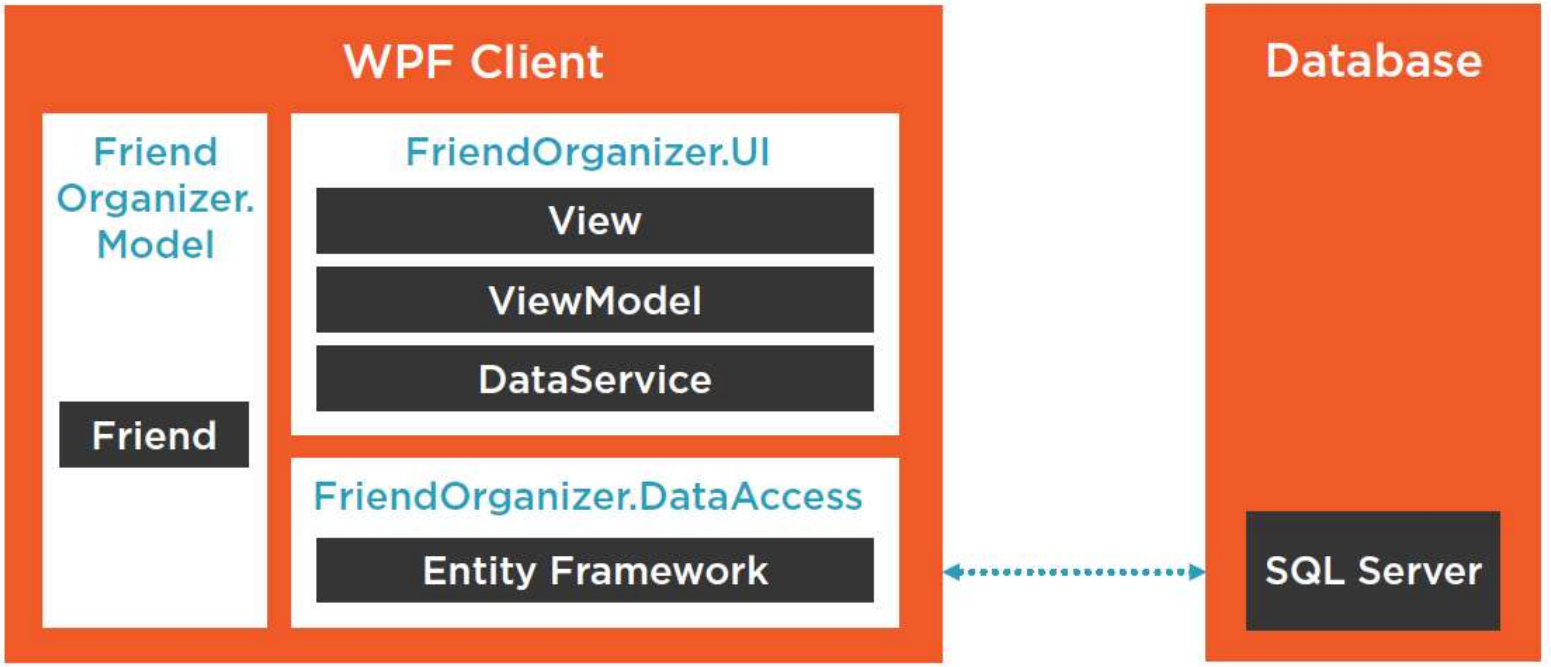
Strati

View: binding, commanding, viste sui dati

ViewModel

- *DataContext* generico sull'intera vista o parti logiche
- Aggrega informazioni
- Effettua chiamate asincrone
- Non si occupa degli aspetti grafici

Strati



Binding

Sfruttare al massimo il data binding nella View

Proprietà nel ViewModel per

- Esporre il modello
- Mantenere lo stato della view
- Eseguire logiche della view

IDataErrorInfo, *Exception* per mostrare gli errori

icubed

Cos'è MVVM?

MVVM è un acronimo che sta per Model-View-ViewModel

MVVM è un design pattern

- Design pattern **è** una serie di linee guida
- Design pattern **non è** una serie di regole

icubed

Model

Sono i dati o le classi che rappresentano le entità dell'applicazione.

Di solito, non contengono codice platform-specific

- Entità
- API
- Servizi, helper, etc…

View

L’interfaccia grafica visibile all’utente. Il *DataContext* è il ViewModel di riferimento.

- XAML
- XAML con code-behind

ViewModel

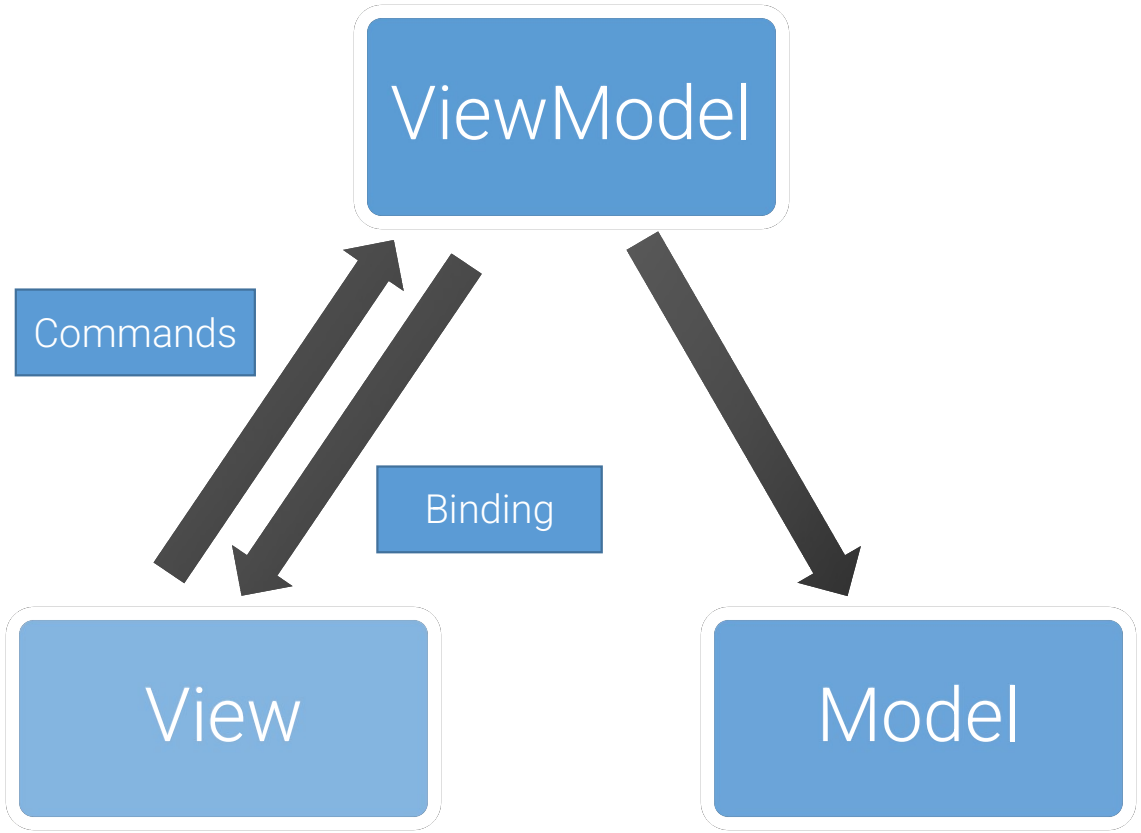
Una classe che contiene tutti i dati che devono essere mostrati nella View e le procedure per recuperare ed elaborare i dati in modo da creare un modello perfetto per la View.

- Stati
- Operazioni
- Codice non dipendente dalla view

La magia di MvvM consente di nascondere la View al ViewModel

icubed

Relazioni



Benefici di MvvM

Basso accoppiamento

Codice testabile (Unit Testing, TDD)

Codice manutenibile

Basso accoppiamento

La View conosce il ViewModel (attraverso il *DataContext*) ma il ViewModel non conosce la View.

Si può sostituire facilmente la View senza intaccare il ViewModel.

Serve in ambito di sviluppo Developer/Designer in quanto i due team possono lavorare in ambiti separati.

Interazione con l’utente

Se voglio far scatenare un evento -> *Command – ICommand*

Se voglio passare dei dati ad un evento -> *CommandParameter*

Se voglio far aggiornare la View con i dati -> *Binding*

Command pattern

ICommand per esporre operazioni

DelegateCommand: *ICommand* basato su delegate

DelegateCommand<T>: *CommandParameter* tipizzato

CommandManager per invalidare i comandi

Messaggi

Eventi

- La view intercetta eventi del VM
- Accoppia troppo al code behind e alla conoscenza del VM

Mediator pattern

- Un mediatore conosciuto da più parti
- Più soggetti che mandano e ricevono messaggi

Binding listener

- Si sfrutta il sistema di binding per monitorare proprietà

Asyncronous pattern

[Operazione]Async

Ogni operazione ha:

- [Operazione]Async: per invocare in asincrono
- [Operazione]Completed: operazione completata
  - Evento scatenato sul thread UI
  - Non dobbiamo preoccuparci se eseguiamo operazioni sulla UI

Implementiamo in classi che verranno usate dalla UI

- Evento con *AsyncCompletedEventArgs*
  - Gestione automatizzata degli errori
  - Aggiungere l'eventuale risultato

Se .NET Framework >=4.5, si può evitare ed usare il pattern *awaitable*

icubed

Async/await & MultiThreading

Un Processo, es. ciascuna delle applicazioni console che abbiamo creato, ha a disposizione risorse come memoria e Thread ad essa allocati

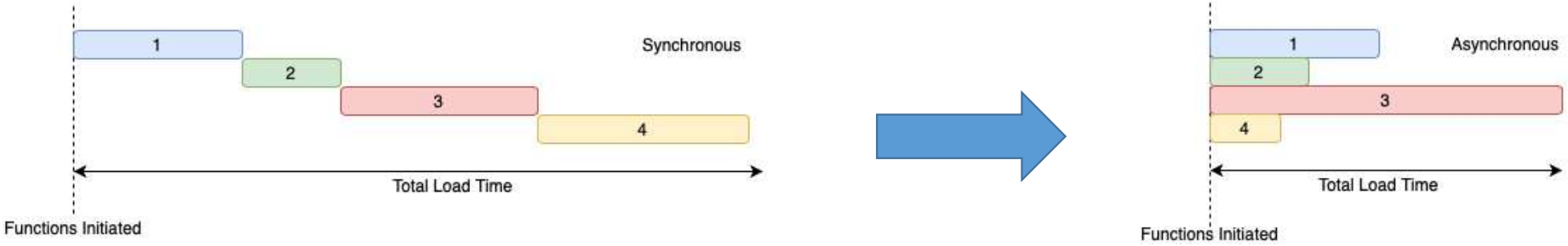
Un Thread esegue il codice, istruzione per istruzione

Per impostazione predefinita, ogni processo ha un solo thread e questo può causare problemi quando sia necessario eseguire più di un'attività (Task) contemporaneamente

Async Await & MultiThreading

Esecuzione di codice Asincrono

Async Await



icubed

Async/await

Per effettuare e semplificare le chiamate asincrone

Nuove parole chiave introdotte con C# 5

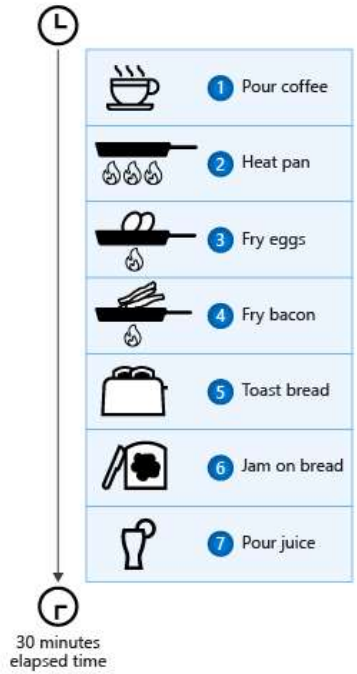
- Async: gestisce la funzione in asincrono
- Await: attende un'operazione asincrona

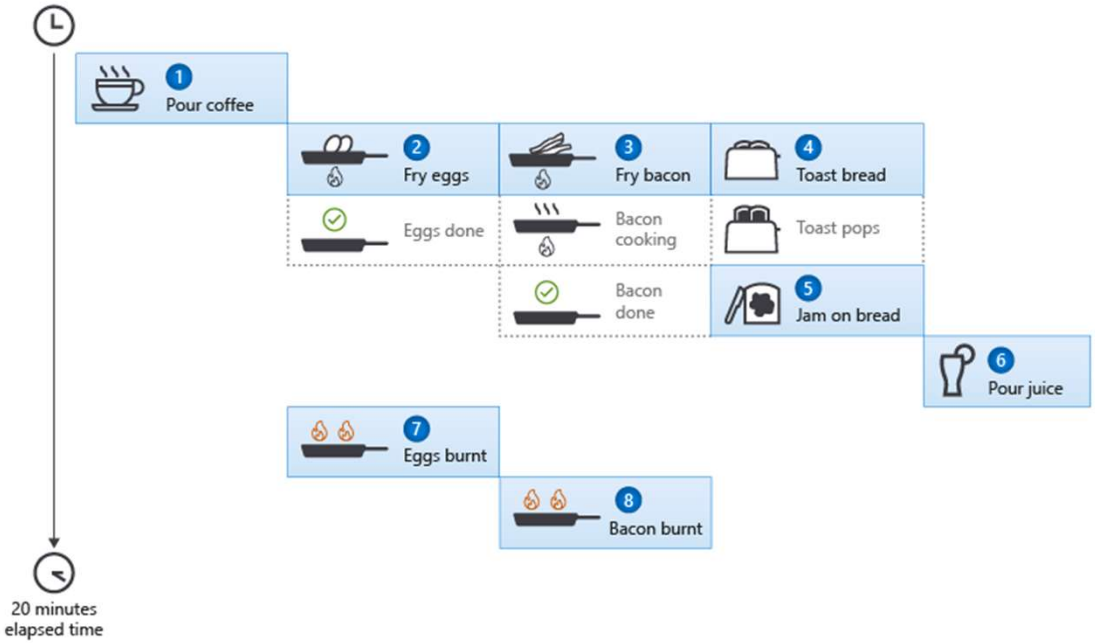
Scriviamo codice come se fosse «sincrono»

Tutto gestito dal compilatore

icubed

Demo – Versione sincrona





icubed