Hola Jairo.

Encantat de trobar-nos al debat del meu TFC. Espero que t'hagi agradat el meu projecte.

Com he intentat explicar a al memòria del projecte l'he plantejat com si es tractés d'una gran aplicació real i per tant he aplicat els patrons arquitectònics i practiques mes adequats i les tecnologies necessàries per implementar-los.

Els objectius principals han estat:

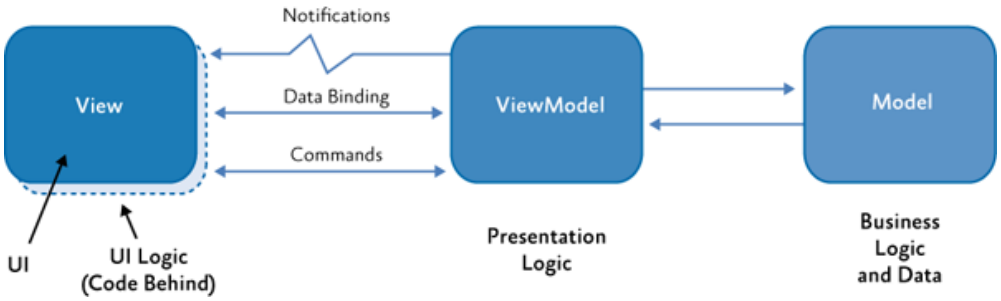
- La separació de conceptes i responsabilitats, atenent al principi de responsabilitat única.
- L'acoblament feble entre components.
- Facilitar la reutilització de components i mètodes.
- Facilitar els tests de proves.

Abans de fer servir cada tecnologia o patró vaig fer una prova de concepte desenvolupant una petita aplicació per provar la implementació. Això em va facilitar molt entendre be els conceptes i evitar problemes mes grans abans d'integrar –ho en el sistema real que estava desenvolupant on les dependències entre components complicarien la resolució d'errors.

A continuació explicaré quina ha estat la meva experiència i conclusió respecte a cada un dels pilars bàsics del meu sistema en quant patrons arquitectònics i tecnologia:

Capa Client Aplicació d'Escriptori:

WPF
Patró de disseny MVVM Model-View-ViewModel per separar el desenvolupament de la interfície d'usuari del desenvolupament de la lògica associada.
Framework Caliburn Micro v1.4



Vaig perdre molts dies prenent contacte i provant diferents frameworks per implementar MVVM i fins i tot vaig provar de implementar-ho directament.

Vaig provar els mes coneguts com *PRISM* massa pesat i no aplicable a la part mòbil, *MVVM-Light de Galasoft*, que vaig veure que no facilitava tant la feina com el que finalment vaig triar, *Caliburn Micro v1.4* que ha estat una agradable sorpresa.

Caliburn Micro es molt mes potent del que pensava. Te la característica de que enllaça directament els controls de la Vista amb les propietats del *View-Model* sense la utilització de "*binding*" explícit si tenen el mateix nom com es veu al següent exemple:

```
<TextBox Name="DocumentId"/>
```

Equival a

```
<TextBox Name="DocumentId" Text="{Binding DocumentId,
Mode=TwoWay, ValidatesOnDataErrors=True}" />
```

Amb els controls que executen accions com els botons funciona d'igual manera. Només cal que el control tingui el mateix nom que el comandament que es vol executar.

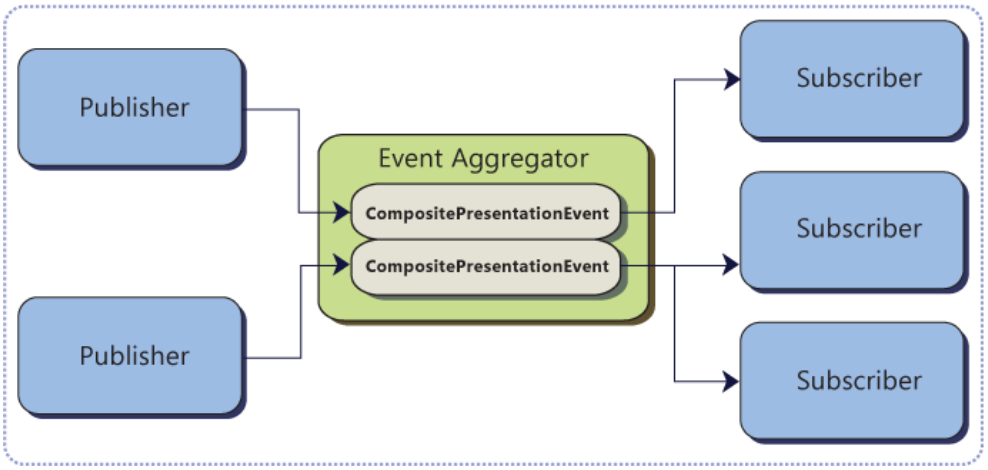
```
<Button Name="Save" Content="D'Acord" Style="{StaticResource
BlueButton}" Width="80" />
```

Aquesta línia d'exemple permet executar el comandament "Save" del ViewModel.
Respecte als comandaments també he trobat molt útil la implementació del mètode Can... , per exemple CanSave vinculada al comandament Save, que conté les expressions booleanes per decidir si un comandament està actiu.

```
public bool CanSave
{
       get
       {
           bool isValid = Expresió booleana;
           return isValid;
       }
}

public void Save()
{
       Accions;
}
```

Les classes pares per fer servir com a base per al ViewModel, com Conductor o Screen també m'han semblat molt útils.
L'agregador d'esdeveninemts, *Event Aggregator*, ha estat un component imprescindible per centralitzar el pas de missatges entre vistes. Aquest component que centralitza la comunicació d'esdeveniments, els publicadors i els subscriptors es una eina molt potent.



La part de navegació, integració d'unes vistes dins d'altres (*nesting*) i de pas de dades i esdeveniments entre vistes va ser de lluny el mes complicat. Finalment vaig fer una prova de concepte específica que reproduïa l'esquema volgut de vistes enllaçades, contenidor inicial (*Shell*), barres de controls, contenidors de formularis, pas de dades i esdeveniments amb l' Event-aggregator de Caliburn , i vaig poder entendre com funcionava i trobar un model base per la meva aplicació.

Amb l'Event aggregator vaig resoldre el problema de fer que una vista reaccione a les accions executades des de un altre.

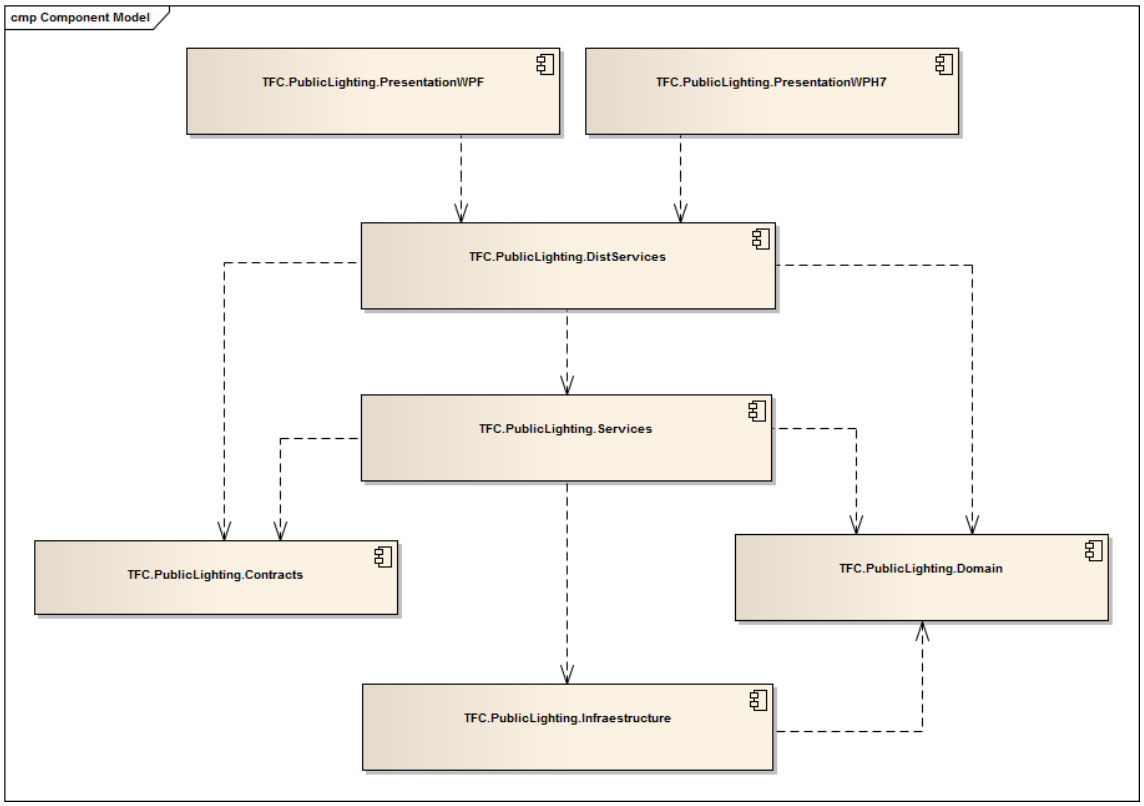
Tot i que Caliburn Micro te un sistema de *Caché* per les vistes, vaig decidir implementar el meu propi a mida per entendre'l millor i tenir nomes la funcionalitat desitjada. Aquest *Caché*, ajudat per alguns mètodes als View-Models em garanteix la persistència de les dades dels Models a la memòria que faig servir per recarregar la vista al tornar-hi, si no es vol una vista nova, o per passar dades d'una vista a un altre.

MVVM i Caliburn, al principi, quan es tot nou, dona una mica de feina fins a tenir clar el model, però després es molt gratificant, quan es veu que tot funciona i es poden reutilitzar classes i mètodes per construir noves interfícies.

Sense dubte ha estat la part que mes m'ha agradat i amb la que em va costar mes començar. Estic content del resultat obtingut amb les interfícies gràfiques de l'aplicació i de com MVVM i Caliburn m'han ajudat.

Arquitectura:

N-capes amb orientació a Domini.



Amb l'arquitectura no he tingut grans problemes. Tenia clars els conceptes tot i que no havia implementat mai una aplicació sencera.

El fet d'estructurar el sistema en mes de les tres capes habituals, pot ser al principi implica una mica més de feina, però després de seguida es veuen els beneficis.
Es molt fàcil substituir uns components per d'altres i fer proves amb distintes versions.

Vaig tenir un problema estúpid de principiant que un dia em va fer perdre unes hores consistent en que els meus clients semblaven no reconèixer els nous objectes DTOs que creava al servidor.
L'error havia estat no haver refrescat el *Service reference* al servei WCF.

Els objectes DTOs els vaig crear de tal manera que directament em serveixen como Models per als ViewModels i les capes clients els reben a través del *Service reference.*

Una solució mes elegant i efectiva seria potser referenciar des dels clients directament el component *TFC.PublicLighting.Contracts* que conté els DTOs.

Respecte als objectes de domini, fins a les últimes versions del sistema, consistien en les entitats generades per les plantilles que es fan servir per *l'Entity Framework* i residien a la capa d'Infraestructura. Aquests objectes no eren agnòstics a la persistència i no es trobaven a la capa de Domini.

Així que vaig millorar l'arquitectura fent servir objectes POCO auto generats per la plantilla *EF 4.x POCO Entity Generator for C# extension* i que si son agnòstics a la persistència. Aquests autèntics objectes de domini si es troben a la capa de Domini.

Capa Infraestructura i accés a dades:

ADO .NET Entity Framework i expresions LINQ per fer les consultes.
SQL Server 2008 R2.

Amb aquesta tecnologia m'he sentit molt còmode des del principi i el fet de treballar amb un model conceptual de les dades i fer consultes amb expressions LINQ redueix el codi i el fa mes clar i mantenible.

Segons avançava al projecte m'he vist obligat en diverses ocasions a refer l'estructura de la base de dades, les taules i les seves relacions.
Això m'ha obligat a refer el model d'*Entity Framework* de nou, tot i que es fa molt ràpid. Diferents intents d'actualitzar el model no m'han funcionat.

Capa Client Aplicació Mòbil:

WPhone 7.1
Silverlight

Per aquesta part del sistema client no he notat gran diferència amb treballar amb WPF ja que Silverlight es pràcticament igual.

Degut a que la lògica es molt mes senzilla i que Windows Phone facilita la navegació entre vistes i hi ha una quantitat petita de controls no vaig trobar necessari fer servir el patró MVVM tot i que *Caliburn* serveix per aplicacions en aquest entorn.

En conclusió crec que ha quedat clar, si se'm permet el to informal, que m'he tornat un admirador de MVVM i Caliburn Micro.

Una cordial salutació.

Toni Navarro.