3. Implementarea aplicației

3.2. Implementarea clientului

Pentru implementarea clientului s-a folosit pachetul de librării .NET *Prism* care permite structurarea aplicației pe module folosind componente slab interconectate care pot evolua independent. Aplicația ce reprezintă clientul este structurată pe trei module, după cum urmează:

i. modulul ce conține întreaga logică necesară pentru a permite utilizatorului să efectueze acțiuni care nu încalcă regulile de joc al unui meci de șah, precum și unele informații suplimentare pentru a ajuta utilizatorul să identifice situația actuală a jocului;
ii. un modul separat este rezervat pentru implementarea notației FEN, care are rolul de a converti dintr-un șir de caractere într-o poziție de șah și viceversa;
iii. al treilea modul este reprezentat de cel vizual, al cărui responsabilitate este să notifice modulul logic în momentul în care utilizatorul interacționează cu aplicația și de a desenare a elementelor vizuale și informațiilor ce reprezintă starea curentă a tablei de joc.

Pentru implementarea unui modul se implementează interfața *IModule*, prezentă în namespace-ul Mic*rosoft.Practices.Prism.Modularity*[5], și precizarea unui nume pentru modul(figura 30). Folosirea modulelor permite o ușoară gestionare a tipurilor folosite în aplicație. Se pune accentul pe folosirea interfețelor, nu se folosesc referințele la tipurile concrete. Folosind un container de injecție putem înregistra în fiecare modul tipul concret necesar. În momentul în care un tip se înregistrează la o interfață se produce o mapare, iar atunci când este inițializat un tip concret containerul de injecție se ocupă automat de furnizarea tipului de date corespunzător respectând maparea făcută(figura 31).

```
    [Module(ModuleName = "GameModule")]
    public class GameModule : IModule
    {
        private void RegisterTypes()
        {
            Container.RegisterType<IEventAggregator,
                                   EventAggregator>();
            Container.RegisterType<IChessSquareViewModel,
                                   ChessSquareViewModel>();
            Container.RegisterType<IChessTableViewModel,
                                   ChessTableViewModel>();
        }
    }
```

Figura 30

Implementarea unui modul

```
    public partial class ChessTableView : UserControl {
// ...
        [Dependency]
        public IChessTableViewModel ViewModel
        {
            set
            {
                DataContext = value;
            }
        }
// ...
}
```

Figura 31

Rezolvarea unui tip de către containerul de injecție

În figura 31 tipul `ChessSquareViewModel` este furnizat automat de către containerul de injecție pe baza înregistrării care se face în modulul *GameModule*.

Un alt aspect al arhitecturii pentru client este respectarea șablonului de proiectare „Stairway”[2] care descrie modul corect de organizare a claselor și interfețelor. Interfețele și implementările lor trebuie să fie în proiecte diferite, ceea ce permite gestionarea independentă a lor, iar clienții acestor funcționalități au nevoie de o singură referință – la librăria de interfețe(figura 32).

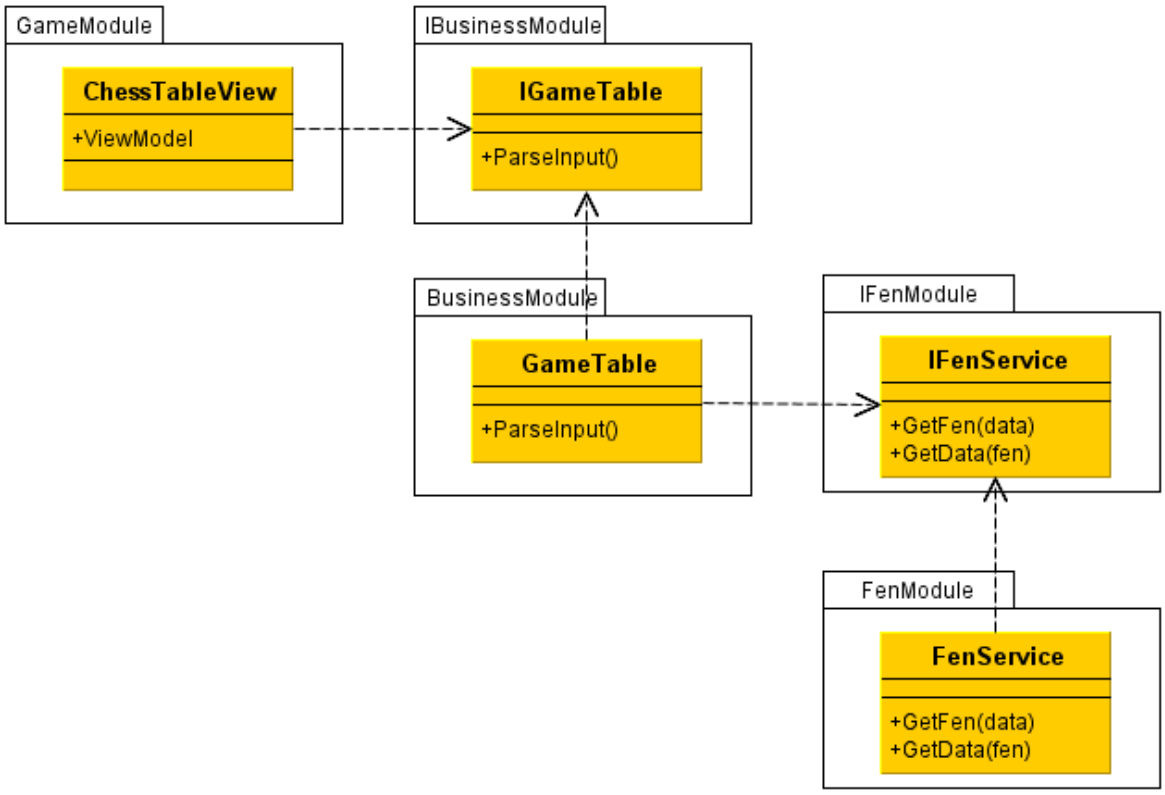

Figura 32

Șablonul de proiectare *Stairway* folosit pentru implementarea clientului

Se observă din figura de mai sus faptul că între componentele ce reprezintă implementări concrete nu există dependințe directe.

Având aceste module implementate, *Prism* oferă trei modalități de încărcare a

acestor module: printr-un fișier de configurare, încărcarea din cod al modulelelor și fișier xaml asemănător celor de configurare. Metoda preferată pentru dezvoltarea clientului este folosirea fișierului de configurare[5] deoarece încărcarea unor module noi nu necesită recompilarea codului(figura 33).

```
  <modules>
    <module assemblyFile="FenService.dll"
moduleType="FenService.FenModule, FenService, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null" moduleName="FenModule"
startupLoaded="true" />
    <module assemblyFile="Chess.Business.ImplementationA.dll"
moduleType="Chess.Business.ImplementationA.ChessImplementationAModu
le, Chess.Business.ImplementationA, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null"
moduleName="ImplementationModule" startupLoaded="true">
      <dependencies>
        <dependency moduleName="FenModule" />
      </dependencies>
    </module>
    <module assemblyFile="Chess.Game.dll"
moduleType="Chess.Game.GameModule, Chess.Game, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null" moduleName="GameModule"
startupLoaded="true">
      <dependencies>
        <dependency moduleName="ImplementationModule" />
      </dependencies>
    </module>
  </modules>
```

Figura 33

Încărcarea modulelor apliției folosind fișierul de configurări

3. Implementarea aplicației

Orice aplicație dezvoltată folosind *Prism* are nevoie de un *Bootstrapper* care inițializează fereastra principală al aplicației și se ocupă de configurarea diferitor componente ce trebuie folosite ulterior.

Fiind pregătită arhitectura sistemului se pot implementa funcționalitățile necesare, principala componentă implicată în implementarea clientului este *ViewModel*-ul *ChessTableViewModel*. Este componenta care face legătura dintre interfața grafică și logica aplicației. Conține proprietăți care sunt folosite pe *view* folosind metode de *binding*[5] și convertori necesari(figura 34).

```
public class ChessTableViewModel : ViewModelBase, IchessTableViewModel {
       public ICommand LoadGameCommand { get; private set; }
       public ICommand SaveGameCommand { get; private set; }
       public ICommand UndoLastMoveCommand { get; private set; }
       public ObservableCollection<IChessSquareViewModel> Squares
              { get; private set; }
}

<UserControl x:Class="Chess.Game.Views.ChessTableView">
       <ListBox Grid.Row="0" Grid.Column="1"
              IsEnabled="{Binding MoveAllowed}" ItemsSource="{Binding Squares}"
              HorizontalAlignment="Left" VerticalAlignment="Top">
       <ListBox.ItemsPanel>
              <ItemsPanelTemplate>
                     <UniformGrid Rows="8" Columns="8" />
              </ItemsPanelTemplate>
       </ListBox.ItemsPanel>
       <ListBox.ItemTemplate>
              <DataTemplate>
              <Button Height="60" Width="60" Margin="-5"
              Background="{Binding Index,
              Converter={StaticResource IndexToBrushConverter}}"
              BorderThickness="{Binding SquareState, Converter={StaticResource
SquareStateToBorderThicknessConverter}}"
              BorderBrush="{Binding SquareState, Converter={StaticResource
SquareStateToBorderColorConverter}}" />
              </DataTemplate>
       </ListBox.ItemTemplate>
       </ListBox>
</UserControl>
```

Figura 34
Implementare *ViewModel-View*