Medii vizuale de programare
Curs 6

Conf. dr.ing. GENGE Béla

Universitatea "Petru Maior", Departamentul de Informatica
Tîrgu Mureș, Romania
bela.genge@ing.upm.ro

Assemblies

- Un assembly reprezintă unitatea primară de dezvoltare într-o aplicație .NET.
- Acestea sunt auto-descriptive și conțin toate informațiile solicitate de CLR privind descrierea și configurarea lor.
- Un assembly este un cod compilat:
  - .EXE: executabil.
  - .DLL: Dynamic Link Library.

Assembly Manifest

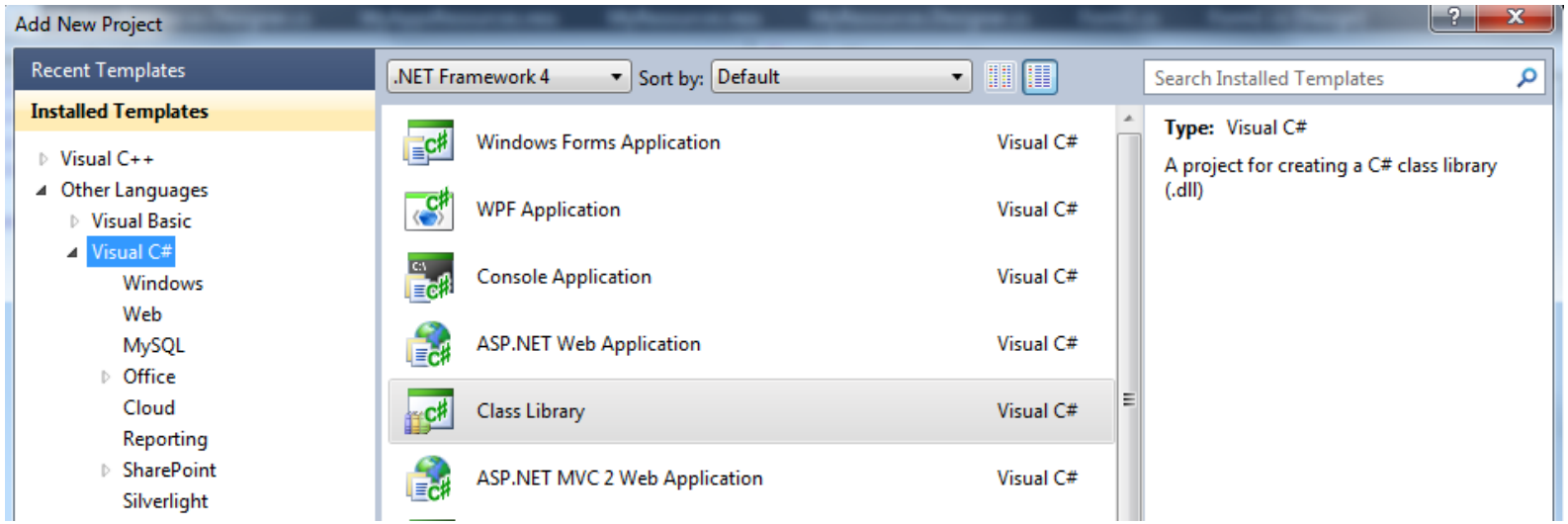
- Fiecare assembly conține un **manifest**: o descriere a assembly-ului (cuprins).
- Assembly-ul este descris în fișierul **AssemblyInfo.cs**

```
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("MyProject")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("MyProject")]
[assembly: AssemblyCopyright("Copyright ©  2015")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

[assembly: ComVisible(false)]
[assembly: Guid("e1664cb9-963f-441a-8238-e5d25e3fc5ea")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

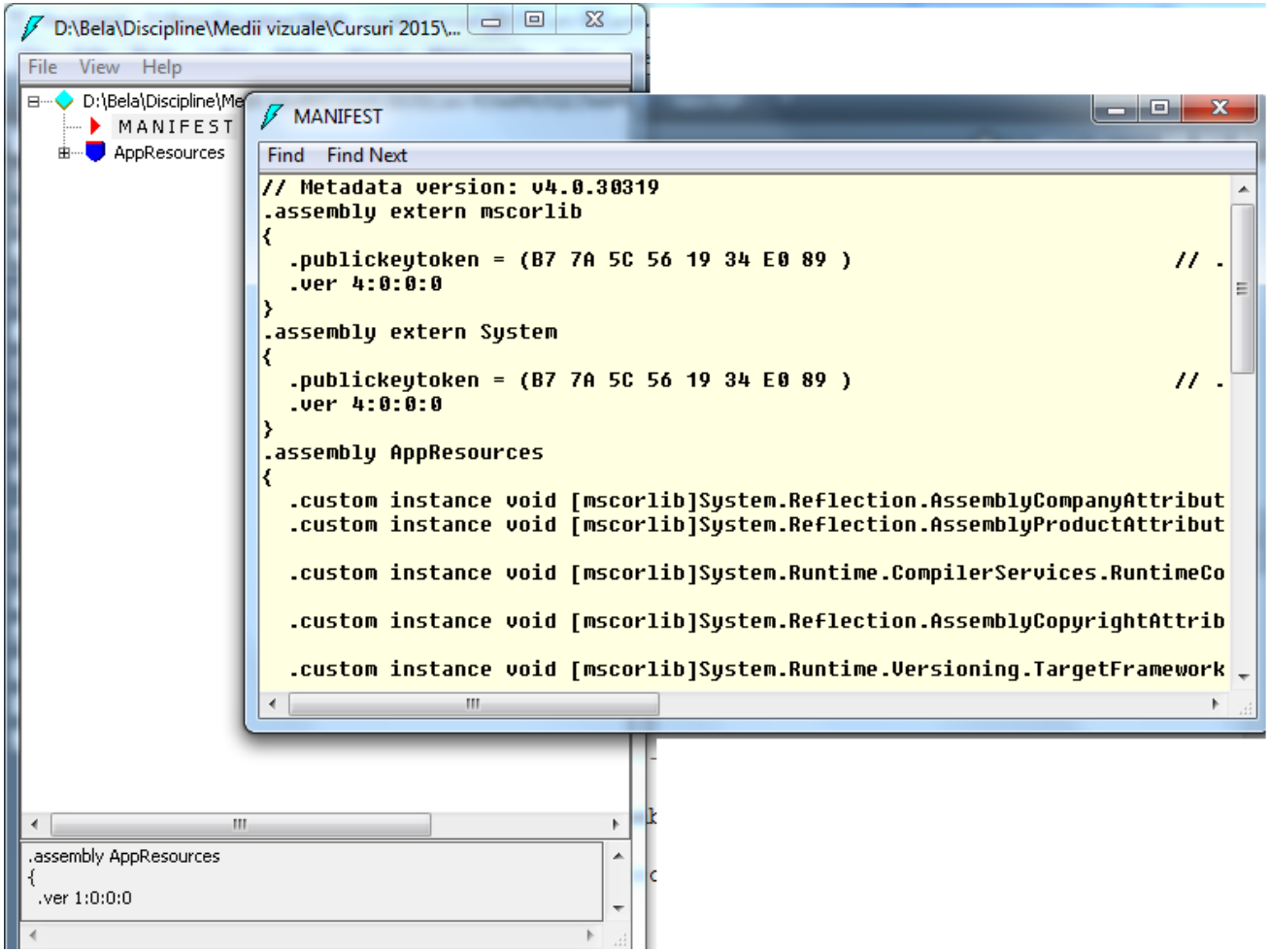
Crearea unui assembly DLL

- În soluția curentă (sau o nouă soluție) se adaugă un nou proiect de tipul **Class Library**.
- Se va crea un nou namespace. Clasele publice vor fi accesibile din afara namespace-ului.



Vizualizare Assembly Manifest

- Tool de la Microsoft: IL Disassembler.



Legarea la un assembly

- Două posibile soluții:
  - Legarea statică (la compilare) a assembly-ului: **Add Reference**.
  - Legarea dinamică (la rulare) a assembly-ului: `System.Reflection`.
- Prin legarea statică (exemplele anterioare cu SQLite și MySQL):
  - Toate tipurile și clasele sunt disponibile la compilare.
  - Se pot realiza verificări de tip și corecta posibile erori la compilare.
  - Încărcarea efectivă a DLL-ului la rulare se realizează la prima accesare.
- Prin legarea dinamică:
  - Assembly-ul nu trebuie să fie disponibil la compilare.
  - Programatorul controlează încărcarea și eliberarea resurselor assembly-ului.
  - Apelul unei metode este mult mai complexă.
  - Eventualele erori de instanțiere/apel al metodelor pot fi detectate la rulare.

Încărcarea prin legarea dinamică

- DLL-ul trebuie să fie lângă .EXE din care este încărcat.
- ... sau, se poate specifica calea completă către DLL.
- ... sau, DLL-ul poate fi înregistrat (instalat) în Global Assembly Cache (slide-urile următoare).
- Pentru exemplul următor vom alege prima variantă:
  - Dacă DLL-ul este dezvoltat în același proiect cu EXE-ul, se poate adăuga la `Post Build Event`:

```
copy MyAssembly.dll
"../../../MyEXE/bin/Debug/MyAssembly.dll"
```

Încărcarea prin legarea dinamica

- Pentru încărcarea dinamică a unui assembly se utilizează clasa `Assembly` din `System.Reflection`.
- Se apelează metoda `Load` și ca parametru se transferă numele assembly-ului.

Încărcare dinamică assembly: MyAssembly

```
System.Reflection.Assembly reflDLL =
System.Reflection.Assembly.Load("MyAssembly");
if (null == reflDLL){
//Tratare eroare.
}
```

Instanțierea prin legarea dinamică

- După ce am încărcat assembly-ul, instanțierea unei clase se realizează prin apelul metodei `CreateInstance()` .
- Ca parametru, se trimite un string reprezentând ierarhia de namespace-uri (delimitate prin '.') și denumirea clasei.

Instanțierea unei clase

```
Object o = reflDLL.CreateInstance("MyNamespace.Class1");
if (null == o){
//Tratare eroare.
}
```

Apelul metodelor prin legarea dinamică

- Presupunem metoda definită în `Class1`:

```
public string getResult(int a, string b)
```

- Clasa `Type` din .NET asigură rezolvarea denumirilor tipurilor în timpul rulării. Apelul unei metode se realizează cu `InvokeMember()`.
- Clasa `Object` pune la dispoziție metoda GetType() pentru a returna o instanță a clasei `Type`.

Apelul metodei

```
string s = (string)o.GetType().InvokeMember("getResult",
System.Reflection.BindingFlags.InvokeMethod,
null,
o,
new object[] { 10, "Test string" });
```

Apelul metodelor prin legarea dinamică

- Începând cu .NET Framework 4.0 se poate utiliza și o altă soluție pentru apelul metodelor prin intermediul tipului **`dynamic`**.
- Dacă o variabilă e declarată `dynamic`, asupra ei NU se aplică verificările statice din timpul compilării.
- Metodele apelate printr-o asemenea variabilă sunt identificate și accesate automat în timpul execuției.

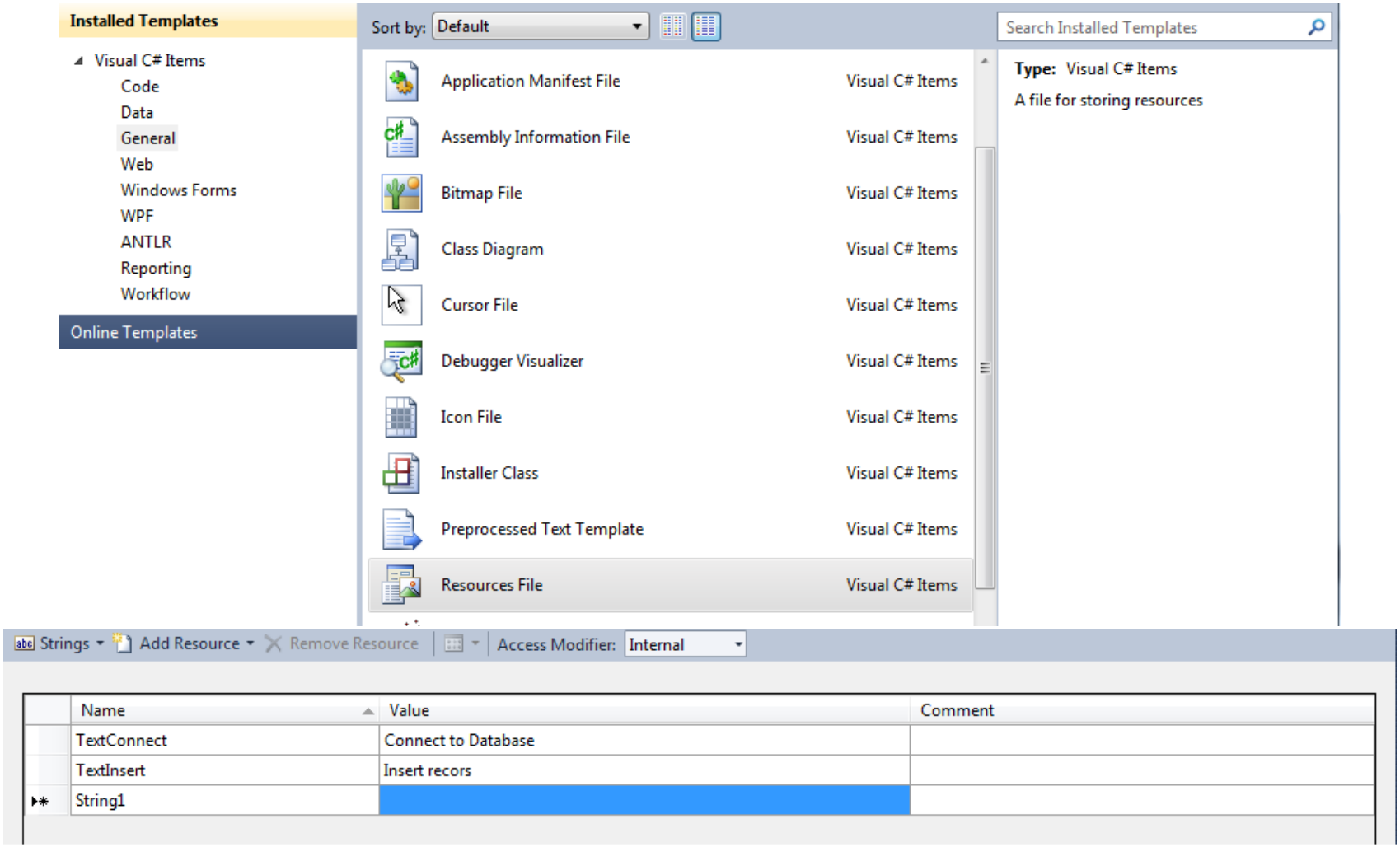
Apelul metodei prin tipul `dynamic`

```
dynamic d = reflDLL.CreateInstance("MyNamespace.Class1");
if (null == d){
//Tratare eroare.
}
string s = d.getResult(10, "Test string");
```

Fișiere pentru resurse

- Sunt date neexecutabile ale aplicației.
- Exemple: șiruri de caractere, imagini.
- Încărcarea textelor vizibile în interfața utilizator dintr-un fișier resursă permite modificarea textelor fără recompilare.
- Avantaje imediate:
  - Suport pentru aplicații în mai multe limbi.
  - Partajarea resurselor în assembly-uri facilitează administrarea unitară și instalrea de update-uri.
- Fișierele resursă pot fi:
  - Fișiere binare de tipul **.resources**
  - Fișiere XML de tipul **.resx**
- Fișierele resursă pot fi compilate în Assembly-uri în mod separat.

Fișiere pentru resurse



Fișiere pentru resurse - accesare

- Accesarea resurselor declarate într-un fișier resursă se realizează prin intermediul `ResourceManager` din `System.Resources`.
- Metoda `GetString()` din `ResourceManager` primește ca parametru un string (denumirea resursei string) și returnează un string reprezentând valoarea resursei.

Fișiere pentru resurse - accesare

Accesarea din același assembly

```
ResourceManager resManager = new
ResourceManager("MyNamespace.MyAppsResources",
this.GetType().Assembly);
string s = resManager.GetString("TextConnect").
```

Accesarea dintr-un assembly diferit

```
System.Reflection.Assembly reflDLL =
System.Reflection.Assembly.Load("MyAssembly");
ResourceManager resManager = new
ResourceManager("AssemblyNamespace.MyAppsResources",
reflDLL);
string s = resManager.GetString("TextConnect").
```

Partajarea assembly-urilor

- Assembly-urile pot fi private sau partajate.
- Assembly-urile private sunt de regulă accesate dintr-o singură soluție. Mai multe proiecte pot utiliza același DLL.
- Pentru ca un assembly să poate fi utilizat de orice aplicație de pe un sistem, acesta trebuie instalat în *Global Assembly Cache*. Avantaje:
  - Securitate: assembly-urile pot fi instalate doar dacă sunt asignate un **strong name**; instalarea se realizează în directorul sistemului de operare unde de regulă drepturile de scriere sunt limitate.
  - Administrare centralizată a tuturor assembly-urilor.
  - Versionare: mai multe versiuni pot fi instalate.

Partajarea assembly-urilor

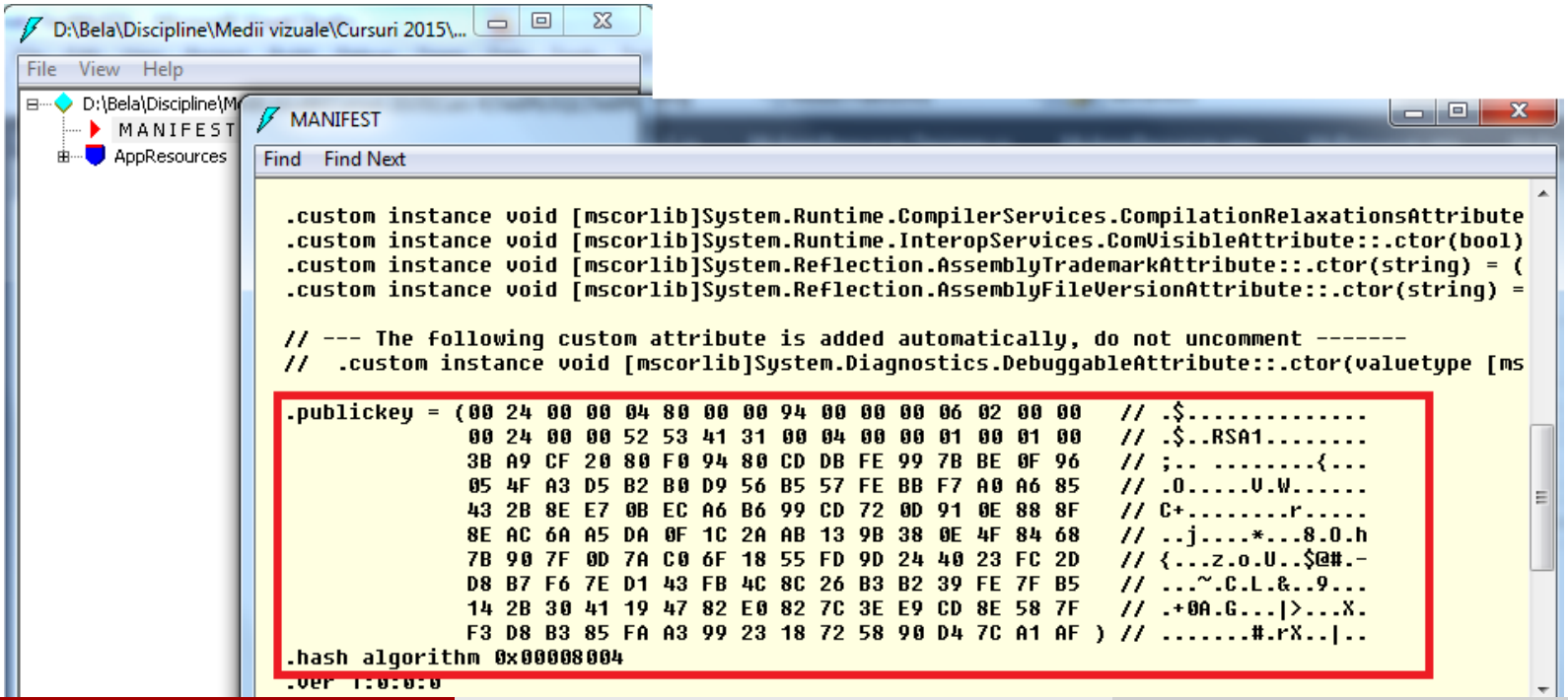
- Un **strong name** este un nume care garantează identitatea unică a unui assembly.
- Acesta include informații: denumire assembly, versiune și o **pereche de chei publica/privata**.
- Informațiile sunt **criptate** folosind cheia privată: criptare cu cheia privată.
- Numai dezvoltatorul aplicației deține cheia privată.
- Verificarea semnăturii se realizează cu cheia publică.

Strong Name Tool

- Microsoft sn.exe.
- Disponibil în Microsoft SDK: C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin
- Generare chei: sn -k MyAssemblyKey.snk
- Extragere cheie publică: sn -p MyAssemblyKey.snk MyAssemblyPK.publickey
- Vizualizare cheie publică: sn -tp MyAssemblyPK.publickey
- Din MS Visual studio: Project properties, Signing, Sign the assembly:
  - Opțiunea New va genera un fișier .pfx - Personal Information Exchange, care e mai mult decât un SNK: conține cheile public/privat, un Certificat X.509, toate criptate cu o parolă.

Verificare dacă un Assembly este Strong Named

- Se deschide DLL-ul cu IL Disassembler.
- Trebuie să includă informații legate de cheia publică.
- Atenție!
  - Semnarea pentru a obține un assembly Strong Named cu un SNK duce responsabilitatea distribuirii cheilor la utilizator.
  - .pfx conține un certificat generat automat, dar care nu este recunoscut! Iarăși, responsabilitatea transferului cheii publice este la utilizator.

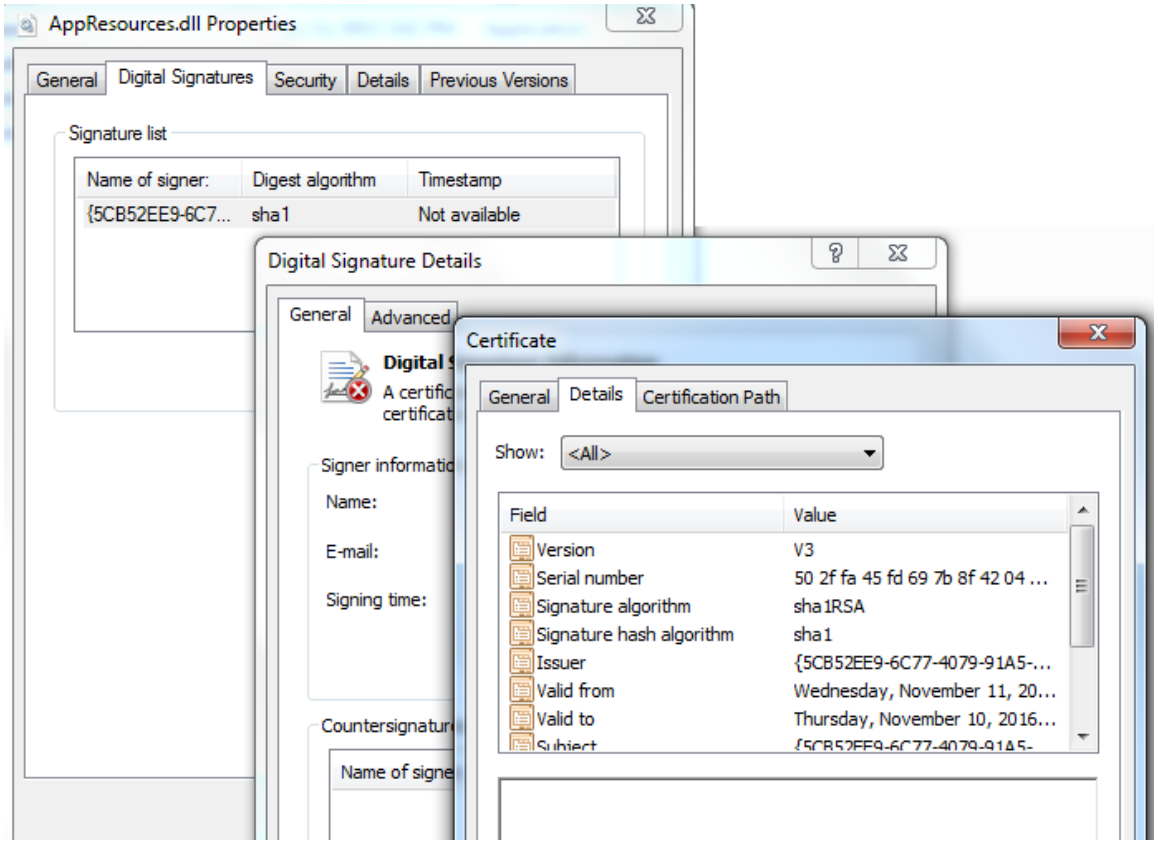


Semnarea digitală a unei aplicații

- Semnătura digitală înseamnă aplicarea unei semnături digitale asupra assembly-ului cu o cheie publică dintr-un certificat digital.
- SN.exe este pentru assembly-uri .NET, dar `signtool.exe` (din Windows SDK) poate semna digital orice aplicație (DLL sau EXE).
- Certificatul poate fi extras dintr-un fișier .pfx, dar aceasta NU este securitate. Pentru securitate trebuie utilizat un certificat EMIS de o AUTORITATE DE ÎNCREDERE.

Semnarea digitală a unei aplicații

signtool.exe sign /f <path-to-pfx-file>MyKeyFile.pfx /p <password-of-pfx-file> <path-to-dll>MyAssembly.dll



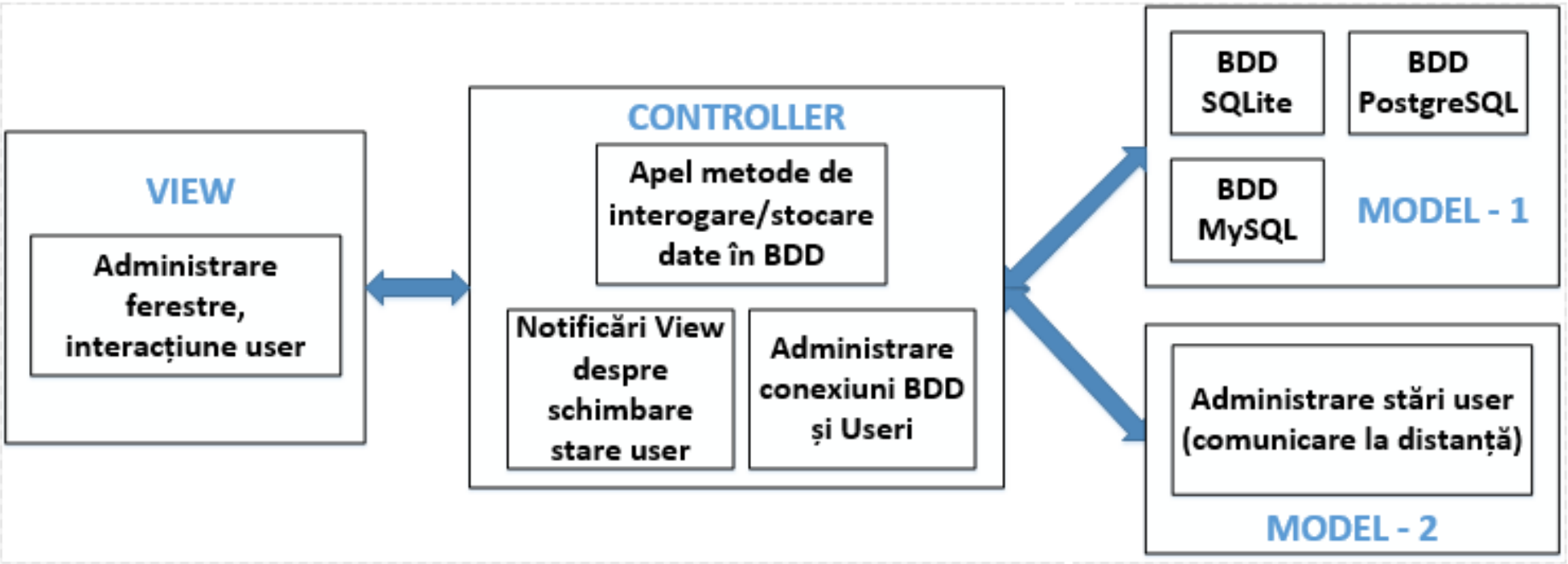
Software design pattern

- Soluție re-utilizabilă pentru proiectarea aplicațiilor software.
- Exemplu de design pattern: singleton.
- Design pattern des utilizat astăzi: Model-View-Controller.

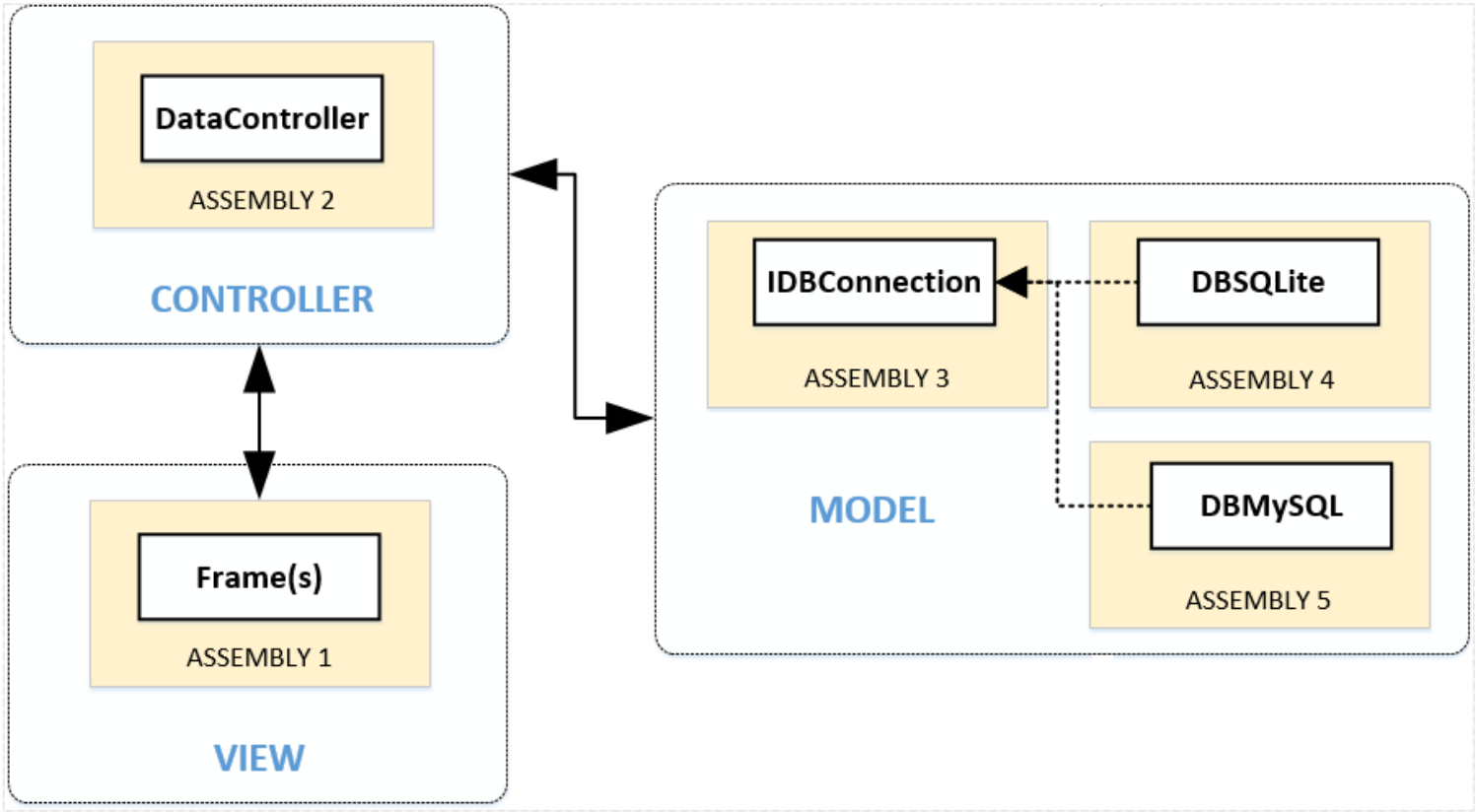
Model View Controller - MVC

- MVC: design pattern ce asigură separarea celor 3 dimensiuni ale unei aplicații.
- Model: administrează datele, starea și comportamentul aplicației.
- View: administrează mecanismele de vizualizare a aplicației.
- Controller: intermediază interacțiunea dintre Model și View (și vice-versa).

Exemplu de aplicație bazat pe MVC



Model View Controller + Assemblies



- Componentele MVC pot fi structurate în assembly-uri diferite.
- Avantaje: vezi cele amintite la assembly-uri + modularizarea implementării + facilitarea muncii în echipă și dezvoltarea proiectelor mari.