Introducció a .NET

Jordi Ceballos Villach

PID_00145166

Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

La plataforma .NET de Microsoft és actualment una de les principals plataformes de desenvolupament d'aplicacions, tant d'escriptori com per a entorns web o dispositius mòbils.

Aquest mòdul pretén donar una visió general de .NET, i també del conjunt de tecnologies que hi ha entorn seu, com ara ADO.NET, WinForms, ASP.NET, WPF, WCF, etc., amb l'objectiu d'oferir un punt de vista general de la plataforma, i donar a conèixer les possibilitats que ofereix.

Tots els exemples i captures de pantalla d'aquesta obra s'han obtingut de Visual Studio 2008, que és la versió disponible de l'IDE de Microsoft en el moment de redactar aquest mòdul.

Objectius

Amb l'estudi d'aquests materials didàctics, assolireu els objectius següents:

1. Obtenir una visió general sobre .NET, tant des del punt de vista de les tecnologies i productes que la componen, com dels avantatges i inconvenients respecte a les tecnologies precedents.

2. Aprendre la sintaxi bàsica del llenguatge C#.

3. Conèixer les principals classes disponibles a la biblioteca de classes .NET Framework, que utilitzarem comunament en els nostres desenvolupaments, tant si es tracta d'aplicacions d'escriptori, com de web o d'altres.

4. Conèixer ADO.NET i LINQ, que ens permeten accedir a les fonts de dades (bases de dades, arxius XML, etc.).

5. Conèixer les possibilitats de les tecnologies Windows Forms, ASP.NET i WPF, que ens permeten crear les interfícies d'usuari de les nostres aplicacions.

6. Obtenir un coneixement bàsic sobre el desenvolupament d'aplicacions per a dispositius mòbils.

1. La plataforma .NET

En aquest apartat s'introdueix la plataforma .NET, la seva arquitectura general, i també els productes i tecnologies que la componen.

1.1. Orígens i característiques de .NET

L'any 2000 Microsoft va presentar la plataforma .NET, amb l'objectiu de fer front a les noves tendències de la indústria del programari i a la dura competència de la plataforma Java de Sun.

Plataforma

Una plataforma és un conjunt de tecnologies, juntament amb un entorn integrat de desenvolupament (IDE) que permeten desenvolupar aplicacions.

.NET és una plataforma per al desenvolupament d'aplicacions que integra diverses tecnologies que han anat apareixent en els últims anys com ASP.NET, ADO.NET, LINQ, WPF, Silverlight, etc., juntament amb el potent entorn integrat de desenvolupament Visual Studio, que permet desenvolupar múltiples tipus d'aplicacions.

Per exemple, podem desenvolupar les aplicacions següents:

- Aplicacions de línia d'ordres.
- Serveis de Windows.
- Aplicacions d'escriptori amb Windows Forms o WPF.
- Aplicacions web amb el *framework* ASP.NET, o Silverlight.
- Aplicacions distribuïdes SOA mitjançant serveis web.
- Aplicacions per a dispositius mòbils amb Windows Mobile.

Framework

Un *framework* és un conjunt de biblioteques, compiladors, llenguatges, etc., que faciliten el desenvolupament d'aplicacions per a una plataforma determinada.

Microsoft només ofereix suport .NET per a sistemes operatius Windows i per a les noves generacions de dispositius mòbils. Pel que fa a la resta de plataformes, el projecte Mono[1] (dut a terme per l'empresa Novell) ha creat una implementació de codi obert de .NET, que actualment ofereix suport complet per a Linux i Windows, i suport parcial per a altres sistemes operatius com, per exemple, MacOS.

(1)Més informació sobre el projecte Mono a www.mono-project.com.

Els elements principals de la plataforma .NET són:

- **.NET Framework**: és el nucli de la plataforma, i ofereix la infraestructura necessària per a desenvolupar i executar aplicacions .NET.

- **Visual Studio i Microsoft Expression**: conformen l'entorn de desenvolupament de Microsoft, que permet desenvolupar qualsevol tipus d'aplicació .NET (d'escriptori, web, per a dispositius mòbils, etc.). A Visual Studio, el programador pot triar indistintament entre diversos llenguatges com, per exemple, C# o Visual Basic .NET, i en tots ells podem fer exactament el mateix, amb la qual cosa sovint escollir-ne un depèn simplement de les preferències personals de cada programador.

Preferències de programari

A causa de la similitud entre llenguatges, sovint els programadors amb experiència en Java escullen programar en C#, mentre que els programadors amb experiència en Visual Basic prefereixen majoritàriament Visual Basic .NET.

1.1.1. Avantatges i inconvenients de .NET

Els principals avantatges de .NET són els següents:

- **Desenvolupament fàcil d'aplicacions**: si les comparem amb l'API Win32 o les MFC, les classes del .NET Framework són més senzilles i completes.

- **Millora de la infraestructura de components**: la infraestructura de components anterior que es va llançar el 1993 (components COM) tenia alguns inconvenients (s'havien d'identificar de manera única, era necessari registrar-los, etc.).

- **Suport de múltiples llenguatges**: .NET no solament ofereix independència del llenguatge (ja ho oferia COM), sinó també integració entre llenguatges. Per exemple, podem crear una classe derivada d'una altra, independentment del llenguatge en el qual hagi estat desenvolupada. Els llenguatges més utilitzats de la plataforma .NET són C# i Visual Basic .NET, tot i que n'hi ha molts d'altres.

Altres llenguatges

Hi ha multitud de llenguatges addicionals com, per exemple, C++, F#, Cobol, Eiffel, Perl, PHP, Python o Ruby.

- **Desplegament senzill d'aplicacions**: .NET torna a les instal·lacions d'impacte zero sobre el sistema, en què només cal copiar una carpeta amb els arxius de l'aplicació per a "instal·lar-la". Tot i que és possible, la majoria d'aplicacions .NET no fan servir el registre de Windows, i desen la seva configuració en arxius XML.

- **Solució a l'infern de les DLL**: permet tenir diferents versions d'una DLL alhora, i cada aplicació carrega exactament la versió que necessita.

L'infern de les DLL

"DLL Hell" va ser durant molts anys un greu problema de Windows, ja que no permetia tenir alhora diferents versions d'una mateixa DLL, i això provocava que certs programes no funcionessin, ja que requerien una versió concreta de DLL.

Pel que fa als inconvenients de .NET, podem destacar els següents:

- **Reduït suport multiplataforma**: Microsoft només ofereix suport per a entorns Windows. El projecte Mono[2], liderat per Miguel de Icaza, ha portat .NET a altres plataformes com Linux o Mac OS X, però el seu suport és limitat.

[(2)]Més informació sobre el projecte Mono a www.mono-project.com.

- **Baix rendiment**: a causa que el codi .NET és en part interpretat, el rendiment és més baix si el comparem amb altres entorns com ara C/C++, que són purament compilats. De fet, per a ser precisos, el codi .NET és,

en primer lloc, compilat per Visual Studio durant el desenvolupament, i posteriorment interpretat pel Common Language Runtime en el moment d'executar-lo.

- **Decompilació**[3]: igual que passa amb Java, les aplicacions .NET contenen la informació necessària que permetria a un *hacker* recuperar el codi font del programa a partir dels fitxers compilats. Per a evitar-ho, podem aplicar tècniques d'ofuscació sobre el codi font, de manera que el seu comportament continua essent el mateix, però com que el codi està ofuscat, compliquem la reenginyeria inversa de l'aplicació.

(3)Consulteu el descompilador Salamander de RemoteSoft.

1.1.2. Evolució de .NET

Des de l'aparició de la primera versió estable de .NET el 2002, Microsoft ha continuat afegint funcionalitats a la plataforma i millorant-ne les eines de desenvolupament.

A continuació, veurem les diferents versions de .NET que hi ha:

a) .NET Framework 1.0: la primera versió del .NET Framework va aparèixer el 2002, juntament amb Visual Studio .NET 2002, el nou entorn de desenvolupament de Microsoft.

b) .NET Framework 1.1: la versió 1.1 apareix el 2003, juntament amb Visual Studio .NET 2003 i el sistema operatiu Windows Server 2003. Per primera vegada apareix .NET Compact Framework, que és una versió reduïda del .NET Framework, dissenyada per a ser executada en dispositius mòbils.

c) .NET Framework 2.0: apareix el 2005, juntament amb Visual Studio 2005 (la paraula *.NET* desapareix del nom del producte) i SQL Server 2005 (la nova versió del motor de bases de dades de Microsoft, 5 anys més tard). Aquesta versió inclou canvis substancials en els llenguatges .NET, com els tipus genèrics o els tipus abstractes. També apareix una segona versió del .NET Compact Framework.

d) .NET Framework 3.0: apareix el 2006, juntament amb Windows Vista. La gran novetat en aquesta versió són les tecnologies següents:

- Windows Presentation Foundation (WPF): facilita el desenvolupament d'interfícies gràfiques avançades, amb gràfics 3D, vídeo, àudio, etc.
- Windows Communication Foundation (WCF): permet el desenvolupament d'aplicacions SOA orientades a serveis.
- Windows Workflow Foundation (WWF): facilita la creació de fluxos de treball que es poden executar des d'una aplicació.

- Windows CardSpace: permet emmagatzemar la identitat digital d'una persona i identificar-la posteriorment.

e) .NET Framework 3.5: apareix al final del 2007, juntament amb Visual Studio 2008, SQL Server 2008 i Windows Server 2008. Aquesta nova versió afegeix LINQ per a l'accés a bases de dades, i també múltiples novetats en l'entorn de desenvolupament (*Javascript intellisense*, possibilitat de desenvolupar per a diferents versions del .NET Framework, etc.).

1.2. Visual Studio

Microsoft Visual Studio és un entorn integrat de desenvolupament (IDE) compartit i únic per a tots els llenguatges .NET. L'entorn proporciona accés a totes les funcionalitats del .NET Framework, així com a moltes altres funcionalitats que fan que el desenvolupament d'aplicacions sigui més àgil.

1.2.1. Evolució de Visual Studio

Visual Studio no és un producte nou; ja existia abans de l'aparició de .NET, per al desenvolupament d'aplicacions mitjançant les tecnologies anteriors. Hi havia diferents versions del producte per a cada un dels llenguatges, bàsicament C++, Visual Basic i J#, a banda de la versió completa, que donava suport a tots els llenguatges en el mateix entorn de treball. L'última versió abans de l'aparició de .NET és la 6.0.

El 2002, amb l'aparició de la versió 1.0 de *.NET*, es va canviar el nom del producte per **Visual Studio .NET 2002**, encara que internament aquesta versió es corresponia amb la versió 7.0.

El 2005 va aparèixer la versió **Visual Studio 2005** (8.0), ja sense la paraula .NET en el nom del producte. Aquesta versió, a banda de proporcionar les noves funcionalitats de la versió 2.0 del .NET Framework, s'integra amb el servidor de bases de dades SQL Server 2005, que va aparèixer alhora.

El 2008, va aparèixer **Visual Studio 2008** (9.0), amb les novetats de la versió 3.5 del .NET Framework, i integrada amb SQL Server 2008.

1.2.2. Edicions de Visual Studio

Visual Studio 2008 es presenta en diferents edicions:

- **Edició express**: edició amb funcionalitats limitades, dissenyada per a desenvolupadors principiants. La podem descarregar i utilitzar gratuïtament, sempre que no sigui amb finalitats lucratives. Hi ha versions express de C#, Visual Basic .NET, C++.NET, Web Developer i SQL Server.

- **Edició estàndard**: permet desenvolupar aplicacions Windows o ASP.NET en qualsevol dels llenguatges de .NET.

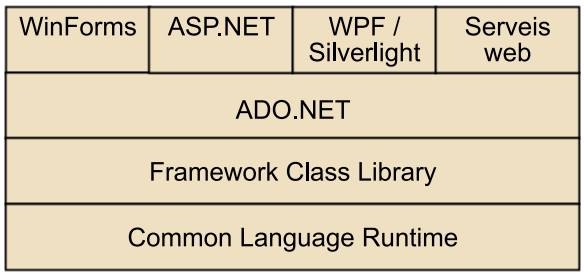
- **Edició professional**: edició pensada per a desenvolupadors professionals. Permet desenvolupar aplicacions per a dispositius mòbils i aplicacions basades en Microsoft Office.

- **Edició Team System**: recomanada per a empreses amb equips de treball. Consta de diverses versions específiques per a cada una de les funcions dins d'un equip de desenvolupament: arquitecte, desenvolupador, *tester*, etc. El producte Visual Studio Team Suite és una versió especial que incorpora totes les funcionalitats dels productes Team System.

1.3. Arquitectura de .NET

El *Common Language Runtime* (CLR) és l'entorn d'execució de .NET, que inclou una màquina virtual, anàloga en molts aspectes a la màquina virtual de Java. El CLR s'encarrega d'oferir l'entorn en què s'executen les aplicacions .NET i, per tant, s'encarrega d'activar els objectes, executar-los, gestionar-ne la memòria, fer comprovacions de seguretat, etc.

Figura 1. Arquitectura de .NET



Per damunt del CLR se situa la Framework Class Library, que amb més de 4.000 classes és una de les biblioteques de classes més grans que hi ha. En el nivell següent, hi ha les classes que permeten l'accés a dades per mitjà d'ADO.NET. I a l'última capa, hi ha les tecnologies per a la creació d'aplicacions, que són les següents:

- **Win Forms**. Desenvolupament d'aplicacions d'escriptori.
- **ASP.NET**. Desenvolupament d'aplicacions web. És l'evolució d'ASP.
- **WPF**. Nova tecnologia per al desenvolupament d'aplicacions d'escriptori.
- **Silverlight**. Subconjunt de WPF destinat al desenvolupament d'aplicacions web. És una tecnologia similar a Flash d'Adobe.
- **Serveis web**. Desenvolupament d'aplicacions distribuïdes.

1.3.1. Compilació i MSIL

En compilar una aplicació .NET obtenim arxius amb extensió .exe o .dll, però no hem de confondre'ns i pensar que estan escrits en codi màquina, sinó en un codi intermedi anomenat MSIL[4]. L'objectiu d'MSIL és el mateix que els *bytecodes* de Java, és a dir, disposar d'un codi intermedi universal (no lligat a cap processador), que pugui ser executat sense problemes en qualsevol sistema que disposi de l'intèrpret corresponent.

(4)De l'anglès, *Microsoft intermediate language*.

En .NET, l'execució està basada en un compilador JIT que, a partir del codi MSIL, va generant el codi nadiu sota demanda, és a dir, compila les funcions a mesura que es necessiten. Com que una mateixa funció pot ser cridada diverses vegades, el compilador JIT, per tal de ser més eficient, emmagatzema el codi nadiu de les funcions que ja ha compilat anteriorment.

Compilació *just in time* (JIT)

La compilació "just a temps" (*just in time*) fa referència a la compilació del codi intermedi MSIL que té lloc en temps d'execució. Es denomina "just a temps" perquè compila les funcions abans mateix que aquestes s'executin.

1.3.2. Assemblatges

Un assemblatge[5] és la unitat mínima d'empaquetament de les aplicacions .NET, és a dir, una aplicació .NET es compon d'un o més assemblatges que acaben essent arxius .dll o .exe.

(5)De l'anglès, *assembly*.

Els assemblatges poden ser privats o públics. En el cas de ser privats, els utilitza una sola aplicació i s'emmagatzemen en el mateix directori que l'aplicació. En canvi, els públics s'instal·len en un directori comú d'assemblatges anomenat *global assembly cache* o GAC, i són accessibles per a qualsevol aplicació .NET instal·lada a la màquina.

A banda del codi MSIL, un assemblatge pot contenir recursos utilitzats per l'aplicació (com, per exemple, imatges o sons), així com diverses metadades que descriuen les classes definides en el mòdul, els mètodes que fa sevir, etc.

2. El llenguatge C#

Tot i que la plataforma .NET permet programar amb múltiples llenguatges, hem seleccionat C# perquè és un dels més representatius i utilitzats de .NET. En aquest apartat, presentem una introducció a la sintaxi del llenguatge C# que, tal com podreu observar, presenta moltes similituds amb llenguatges com C++ i Java.

2.1. Una introducció al llenguatge C#

La millor manera de començar a aprendre un llenguatge de programació nou és mitjançant l'anàlisi d'algun exemple de codi senzill. Per no trencar la tradició, comencem amb el típic *HolaMon*:

```
namespace HolaMon
{
  class HolaMon
  {
    static void Main(string[] args)
    {
      Console.WriteLine ("Hola Món");
    }
  }
}
```

Analitzem els elements del programa anterior:

- **Definició del *namespace***: totes les classes estan incloses dins d'un espai de noms concret, que s'indica dins del codi font mitjançant la instrucció *namespace*.

- **Definició de la classe**: la manera de definir una classe és mitjançant la paraula clau *class*, seguida del nom que volem donar a la classe, i tots els elements i instruccions que pertanyen a la definició de la classe entre claus. S'ha de destacar que a C# tot són classes, amb la qual cosa totes les línies de codi han d'estar associades a alguna classe.

- **El mètode *main***: punt d'entrada a l'aplicació, per on en comença l'execució. Pot rebre un *array* anomenat *args*, que conté els paràmetres passats a l'executable en llançar-ne l'execució.

Els tipus de dades de C# es divideixen en dues categories:

1) Tipus valor

Són els tipus primitius del llenguatge: enters, reals, caràcters, etc. Els tipus valor més usuals són:

Tipus	Descripció
short	Enter amb signe de 16 bits
int	Enter amb signe de 32 bits
long	Enter amb signe de 64 bits
float	Valor real de precisió simple
double	Valor real de precisió doble
char	Caràcter Unicode
bool	Valor booleà (*true*/*false*)

2) Tipus referència

Els tipus referència ens permeten accedir als objectes (classes, interfícies, *arrays*, *strings*, etc.). Els objectes s'emmagatzemen a la memòria *heap* del sistema, i hi accedim a través d'una referència (un punter). Aquests tipus tenen un rendiment més baix que els tipus valor, ja que l'accés als objectes necessita un accés addicional a la memòria *heap*.

2.2. Sintaxi de C#

La sintaxi d'un llenguatge és la definició de les paraules clau, els elements i les combinacions vàlides en aquest llenguatge. A continuació, descrivim de manera resumida la sintaxi de C#.

2.2.1. *Case sensitive*

C# és un llenguatge *case sensitive*, és a dir, que diferencia entre majúscules i minúscules. Per exemple, si escrivim *Class* en lloc de *class*, el compilador donarà un error de sintaxi.

2.2.2. Declaració de variables

En C# hem de declarar totes les variables abans de fer-les servir, i és aconsellable assignar-hi sempre un valor inicial:

```
int prova = 23;
float valor1 = 2.5, valor2 = 25.0;
```

2.2.3. Constants

Una constant és una variable el valor de la qual no pot ser modificat, és a dir, que és només de lectura. Es declaren amb la paraula clau *const*:

```
const int i = 4;
```

2.2.4. *Arrays*

Un *array* permet emmagatzemar un conjunt de dades d'un mateix tipus, a les quals accedim segons la seva posició en l'*array*. En la línia següent, declarem un *array* que emmagatzema 5 enters, i desem un primer valor:

```
int[] elMeuArray = new int[5];
elMeuArray[0] = 10;
```

2.2.5. Comentaris

Els comentaris són imprescindibles si pretenem escriure codi de qualitat. En C# hi ha tres tipus de comentaris:

Comentaris	
D'una sola línia	`// Exemple de comentari`
De múltiples línies	`/* ... */`
	Qualsevol caràcter entre el símbol /* i el símbol */ és considerat com part del comentari, encara que abasti diverses línies.
Comentaris XML	`///`

Lectura recomanada

Recomanem la lectura del llibre *Code Complete*, de Steve McConnell, una guia excel·lent per a escriure codi de qualitat. Conté un capítol interessant dedicat als comentaris.

Incorporem etiquetes o *tags* XML documentant el codi, i així després podem generar un arxiu XML amb tota la documentació del codi:

```
/// <summary> Breu descripció d'una classe</summary>
///<remarks> Exemple d'ús de la classe</remarks>
public class LaMevaClasse() { ... }
```

Una vegada tenim el codi comentat amb etiquetes XML, podem generar una sortida elegant en format HTML o CHM amb l'eina SandCastle de Microsoft.

2.2.6. Visibilitat

Podem restringir l'accés a les dades i mètodes de les nostres classes indicant un dels modificadors d'accés següents:

- ***public***: sense restriccions.
- ***private***: accessible només des de la mateixa classe.

- ***protected***: accessible des de la mateixa classe, i des de classes derivades.
- ***internal***: accessible des del mateix assemblatge.

Exemple:

```
class Prova {
  public int quantitat;
  private bool visible;
}
```

2.2.7. Operadors

A C# podem fer ús dels operadors següents:

Operadors	
Aritmètics	+, –, *, /, % (mòdul)
Lògics	& (AND bit a bit), | (OR bit a bit), ~ (NOT bit a bit)
	&& (AND lògic), || (OR lògic), ! (NOT lògic)
Concatenació de cadenes de caràcters	+
Increment/decrement	++, —
Comparació	==, != (diferent), <, >, <=, >=
Assignació	=, +=, –=, *=, /=, %=, &=, |=, <<=, >>= Les combinacions += –=, *=, etc. permeten assignar a una variable el resultat de dur a terme l'operació indicada. Exemple: *x* += 3 equival a *x* = *x* +3

2.2.8. Enumeracions

Una enumeració és una estructura de dades que permet definir una llista de constants i assignar-hi un nom. Tot seguit, mostrem una enumeració per als dies de la setmana:

```
enum DiaSetmana
{
  Dilluns, Dimarts, Dimecres, Dijous, Divendres, Dissabte, Diumenge
}
```

D'aquesta manera el codi és molt més llegible, ja que en comptes d'utilitzar un enter de l'1 al 7 per a representar el dia, es fa servir una constant i un tipus específic de dades per al dia de la setmana, de manera que s'eviten problemes addicionals com, per exemple, controlar que el valor d'una variable que representi el dia estigui entre 1 i 7.

2.2.9. Estructures

Una estructura conté diversos elements que poden ser de diferents tipus de dades:

```
struct Punt
{
  int x;
  int Y;
  bool visible;
}
```

Les estructures es declaren igual que qualsevol tipus per valor:

```
Punt p;
p.x = 0;
p.y = 0;
```

2.2.10. Control de flux

El llenguatge C# incorpora les sentències de control de flux següents:

- ***if***: sentència condicional. Un exemple típic seria el següent:

```
if (x <= 10)
{
  // Sentències per executar si la condició és certa
}
else
{
  // Sentències per ejecutar si la condició és falsa
}
```

- ***switch***: la instrucció *switch* és una forma específica d'instrucció condicional, en la qual s'avalua una variable, i en funció del seu valor, s'executa un bloc d'instruccions o un altre. Exemple:

```
switch (var)
{
  case 1: // Sentències per executar si var és 1
          break;

  case 2: // Sentències per executar si var és 2
          break;
  ...

  default: // Sentències per executar
```

```
            // en cas que cap de les
            // condicions "case" es compleixin
            break;
}
```

- ***for***: permet executar un bloc de codi un cert nombre de vegades. L'exemple següent executa el bloc d'instruccions 10 vegades:

```
for (int i = 0; i < 10; i++)
{
  // Sentències per executar
}
```

- ***while***: l'exemple següent equival a l'anterior del *for*.

```
int i = 0;
while (i < 10)
{
     // Sentències per executar
     i++;
}
```

- ***do-while***: igual que l'anterior, excepte que la condició s'avalua al final. La diferència fonamental és que, mentre que en un bucle *for* o *while*, si la condició és falsa d'entrada, no s'executa cap iteració, en un bucle *do-while* sempre s'executa com a mínim una iteració.

```
do
{
  // Sentències per executar
}
while (condició);
```

- ***foreach***: permet recórrer tots els elements d'una col·lecció des del primer fins a l'últim. La sintaxi és la següent:

```
foreach (tipus nom_variable in col·lecció)
{
  // Sentències per executar
}
```

Un exemple de col·lecció són els *arrays*:

```
int [] nums = new int [] { 4,2,5,7,3,7,8 };
foreach (int i in nums)
{
  Console.WriteLine (i);
```

```
}
```

2.2.11. Pas de paràmetres

En C# el pas de paràmetres a una funció es pot fer per valor, per referència, o pot ser un paràmetre de sortida.

Pas de paràmetres per valor

En primer lloc, recordem que els tipus de dades que hàgim de passar com a paràmetre poden ser de tipus valor (tipus primitius) o tipus referència (objectes), ja que analitzarem tots dos casos per separat.

En passar un tipus valor com a paràmetre a una funció, es passa per valor, és a dir, el valor de la variable es copia en el mètode, en la variable local representada pel paràmetre. D'aquesta manera, si es modifica el valor de les variables dels paràmetres, no es modifica el valor de les variables que es van passar inicialment com a paràmetre. A l'exemple següent, una vegada executat el mètode *PasDeParametres*, la variable *a* continuarà valent 0.

```
static void PasDeParametres (int param)
{
  param = 5;
}

static void Main(string[] args)
{
  int a = 0;
  PasDeParametres(a);
}
```

En el cas dels tipus referència, el que en realitat es passa com a paràmetre a una funció és la seva adreça de memòria. Per tant, en accedir al paràmetre dins del mètode, es modifica la variable original. Tanmateix, el que no podem fer és canviar l'adreça de memòria a la qual apunta la variable original.

Excepció

L'*string* és una excepció, ja que en passar un *string* com a paràmetre, es copia la cadena, és a dir, es comporta com un tipus valor.

En l'exemple següent, una vegada executat el mètode, *PasDeParametres*, la posició 0 del vector *a* passarà a tenir valor 5, i tot i que en la funció se li assigna un nou *array* amb valors negatius, això no té cap rellevància en sortir de la funció.

```
static void PasDeParametres (int[] param)
{
  param[0] = 5;
  param = new int[4] { -1, -2, -3, -4};
}

static void Main(string[] args)
```

```
{
  int[] a = { 0, 1, 2, 3 };
  PasDeParametres(a);
}
```

Pas de paràmetres per referència

Analitzarem el pas de paràmetres per referència primer amb tipus valor, i tot seguit amb tipus referència.

Per a passar un tipus valor com a paràmetre per referència a una funció, ho indiquem amb la paraula clau *ref*. En aquest cas, la funció rep la direcció de memòria de la variable, amb la qual cosa si la modifica, el canvi té lloc també a la variable original. En l'exemple següent, una vegada executat el mètode, *PasDeParametres* la variable a passarà a valer 5.

```
static void PasDeParametres(ref int param)
{
  param = 5;
}
static void Main(string[] args)
{
  int a = 0;
  PasDeParametres(ref a);
}
```

Per a passar un tipus referència com a paràmetre per referència a una funció, ho indiquem amb la paraula clau *ref*. En aquest cas, la funció rep l'adreça de memòria on està emmagatzemada la variable original, amb la qual cosa podem modificar tant els valors de la variable original, com també fer que la variable original acabi apuntant a una adreça diferent.

En l'exemple següent, una vegada executat el mètode, *PasDeParametres* la posició 0 del vector *a* passarà a tenir valor –1, ja que realment hem fet que apunti a l'adreça de memòria del nou *array*.

```
static void PasDeParametres(ref int[] param)
{
  param[0] = 5;
  param = new int[4] { -1, -2, -3, -4 };
}
static void Main(string[] args)
{
  int[] a = { 0, 1, 2, 3 };
  PasDeParametres(ref a);
  Console.WriteLine(a[0]);
```

```
}
```

Paràmetres de sortida

Finalment, també existeixen els paràmetres de sortida, els quals s'indiquen amb la paraula clau *out*, i només serveixen per a retornar valors en un mètode.

A l'exemple següent, la funció torna els valors $a = 5$ i $b = 10$. En cas que la funció hagués intentat llegir els valors de a i b abans d'assignar-los cap valor, hauríem obtingut un error de compilació.

```
static void PasDeParametres(out int a, out int b)
{
  a = 5;
  b = 10;
}

static void Main(string[] args)
{
  int a, b;
  PasDeParametres(out a, out b);
}
```

2.2.12. Sobrecàrrega de mètodes

La sobrecàrrega de mètodes permet tenir múltiples mètodes amb el mateix nom, encara que amb paràmetres diferents. En fer la crida al mètode, se n'invoca l'un o l'altre en funció dels paràmetres de la crida. Vegem-ne un exemple:

```
public static int prova(int a)
{
  return 2 * a;
}

public static int prova(int a, int b)
{
  return a + b;
}

static void Main(string[] args)
{
  Console.WriteLine(prova(10));
  Console.WriteLine(prova(10, 40));
}
```

2.3. Programació orientada a objectes amb C#

C# és un llenguatge orientat a objectes. Tot seguit, veurem els conceptes bàsics de l'orientació a objectes i la seva utilització en C#.

2.3.1. Definició de classes

En C# podem definir una classe mitjançant la paraula clau *class*:

```
class LaMevaClasse
{
  public int valor; // Membre de dades

  public int calcul() // Membre de funció
  {
    return valor*2;
  }
}
```

En la definició de la classe, especifiquem els membres de dades (variables o propietats) que descriuen l'estat de l'objecte, i un conjunt d'operacions que en defineixen el comportament (funcions o mètodes).

Hi ha la possibilitat de definir una classe en diversos fitxers de codi font, la qual cosa rep el nom de *classes parcials*. Si creem un formulari en Visual Studio, automàticament es crea una classe parcial repartida en dos arxius; en un escrivim el codi associat als esdeveniments, i en l'altre Visual Studio genera automàticament el codi de la interfície d'usuari.

A tall d'exemple, si creem un formulari anomenat *Form1*, obtenim una classe parcial anomenada *Form1* repartida en els arxius:

- **Form1.cs**: en aquesta classe, programarem tot el codi dels esdeveniments del formulari que volem implementar (en carregar la pàgina, en prémer un botó, etc.).

- **Form1.Designer.cs**: mai modificarem aquesta classe manualment, sinó que Visual Studio generarà el codi corresponent al formulari que dissenyem mitjançant Visual Studio.

2.3.2. Objectes

Un objecte és una instància concreta d'una classe. La sintaxi per a la instanciació d'un objecte és la següent:

```
LaMevaClasse obj;
```

```
obj = new LaMevaClasse ();
obj.valor = 3;
```

Si no volem fer més ús d'un objecte, podem assignar-li el valor *null*. D'aquesta manera, el recuperador de memòria de C# detectarà que l'objecte ha deixat de ser referenciat i, per tant, alliberarà la memòria que fa servir.

2.3.3. Propietats

Una propietat permet encapsular una variable juntament amb els mètodes que permeten consultar-ne o modificar-ne el valor. En altres llenguatges, les propietats no existeixen, amb la qual cosa per a cada variable d'una classe s'afegeixen manualment els mètodes *get/set* corresponents.

A l'exemple següent, definim la propietat *Descompte*, el valor de la qual no permetem que pugui ser negatiu:

```
public class LaMevaClasse
{
  private int descompte;
  public int Descompte
  {
    get
    {
      return descompte;
    }
    set
    {
      if (value > 0) descompte = value;
      else descompte = 0;
    }
  }
}
```

Una vegada definida la propietat, utilitzar-la és molt senzill:

```
LaMevaClasse c = new LaMevaClasse();
c.Descompte = 10;
Console.WriteLine(c.Descompte);
```

2.3.4. Construcció d'objectes

Un constructor és el mètode que permet crear objectes d'una classe determinada i assegurar que els objectes s'inicialitzin correctament, amb la qual cosa assegurem la integritat de les instàncies. És habitual sobrecarregar els construc-

tors per a disposar de múltiples constructors en funció del nombre de paràmetres rebuts. En l'exemple següent, la classe té un constructor per defecte (sense paràmetres), d'una banda, i de l'altra, un constructor amb un paràmetre:

```
class LaMevaClasse
{
  public int valor;
  public LaMevaClasse()
  {
    valor = 0;
  }
  public LaMevaClasse(int valor)
  {
    this.valor = valor;
  }
}
```

En l'exemple anterior hem utilitzat la paraula clau *this*, que permet accedir als membres de dades de la pròpia classe. En cas de no haver utilitzat *this*, hauríem d'haver posat un nom diferent del paràmetre del constructor perquè no coincideixi amb el nom de la variable.

En tenir dos constructors, podem crear objectes de dues maneres diferents:

```
LaMevaClasse obj1, obj2;
obj1 = new LaMevaClasse();
obj2 = new LaMevaClasse(10);
```

2.3.5. Destrucció d'objectes

Els mètodes destructors permeten alliberar els recursos que ha utilitzat l'objecte en executar-lo, però que a partir d'un cert moment han deixat de ser utilitzats i, per tant, estan esperant ser alliberats, com passaria per exemple amb una connexió a una base de dades.

Un exemple de destructor seria el següent:

```
~LaMevaClasse()
{
  Console.WriteLine("L'objecte ha estat destruït");
}
```

El caràcter ~

El caràcter ~ normalment no apareix en els teclats espanyols, però podem fer-lo aparèixer prement Alt + 126 (en el teclat numèric).

L'alliberament de recursos té lloc mitjançant el procés de recuperació automàtica de memòria[6], que s'encarregarà, entre d'altres coses, d'executar els destructors dels objectes que no s'hagin d'utilitzar més. Això sí, l'alliberament de recursos no es produeix immediatament. De fet, no podem predir quan s'executarà el *garbage collector* i, fins i tot pot arribar a passar que aquest no s'executi fins que hagi finalitzat l'execució del nostre programa.

[6] De l'anglès, *garbage collection*.

2.3.6. Mètodes estàtics

Un mètode estàtic és aquell que podem invocar sense necessitat de crear prèviament un objecte de la classe que el conté. Com a exemple, mostrem el mètode *Pow* de la classe *Math*:

```
double resultat = Math.Pow(2.0, 8);
```

Un altre exemple de mètode estàtic és el mètode *Main*, que no s'executa sobre cap instància de classe, ja que en ser el primer mètode que s'executa, encara no s'ha creat cap objecte.

2.3.7. Herència

Les classes es poden organitzar en jerarquies que permeten ampliar i reutilitzar les funcionalitats d'unes classes a d'altres. Aquestes jerarquies es creen mitjançant la relació d'herència, que relaciona dues classes: la superclasse i la subclasse, en què una subclasse hereta automàticament tota la funcionalitat de la superclasse.

```
class LaMevaClasseFilla : LaMevaClassePare
{
}
```

Si en definir una classe no n'especifiquem la classe pare, per defecte aquesta classe rep l'herència de la classe *Object*, que conté alguns mètodes comuns a totes les classes, com per exemple *Equals* o *ToString*.

En C# una classe només pot heretar d'una única classe pare. En canvi, hi ha llenguatges que permeten l'herència múltiple, és a dir, que les classes poden heretar de múltiples classes pare alhora.

2.3.8. Interfícies

Una interfície és una mena de plantilla que especifica com han de ser un conjunt de mètodes (indicant-ne els paràmetres i valors de retorn), però no en disposa de la implementació.

Una vegada definida una interfície, hi haurà classes que la implementaran i que, per tant, estaran obligades a implementar tots aquells mètodes indicats a la interfície. De fet, en el nivell de les interfícies no hi ha problemes amb l'herència múltiple, de manera que una classe pot implementar sense problemes múltiples interfícies alhora (en cas d'haver-hi mètodes de les classes pare que coincideixin, no hi ha problema, ja que aquests mètodes només tindran una única implementació que serà la definida en la pròpia classe).

En l'exemple següent, definim la interfície *IProva* i la classe *LaMevaClasse* implementa l'esmentada interfície, amb la qual cosa es veu obligada a implementar el mètode *calcul* (en cas de no fer-ho, el codi no compilaria):

```
public interface IProva
{
  double calcul(double x);
}

public class LaMevaClasse : IProva
{
  public double calcul(double x)
  {
    return Math.Pow(x, 3);
  }
}
```

2.3.9. Sobreescriptura de mètodes

Quan una classe hereta d'una altra, pot sobreescriure els mètodes de la superfície marcats com a virtuals, i en sobreescriure'ls ha d'indicar-ho explícitament amb el modificador *override*. A tall d'exemple, la classe *Object* ofereix el mètode virtual *ToString*, que normalment sobreescrivim en les nostres classes perquè la crida es faci a la nostra pròpia implementació del mètode, i no a la de la classe base. En cas que volguéssim fer la crida al mètode de la classe base, ho faríem amb la sentència *base.ToString*().

En l'exemple següent, creem una classe *Persona* que implementa el seu propi mètode *ToString* i, en el mètode *Main*, es mostrarà per pantalla el nom de la persona. Quan el mètode *Console.WriteLine* no rep com a paràmetre un *String*, automàticament fa una crida al mètode *ToString*() de l'objecte que li hàgim passat com a paràmetre.

```
class Persona
{
  String nom;
  public Persona(String nom)
  {
    this.nom = nom;
```

```
  }
  public override string ToString()
  {
    return nom;
  }
}

class Program
{
  static void Main(string[] args)
  {
    Persona p = new Persona("Lluís");
    Console.WriteLine(p);
  }
}
```

2.3.10. Genèrics

Els tipus genèrics són classes que es poden aplicar sobre diferents tipus de dades. Per a fer-ho utilitzen un tipus de dades comodí, que pot prendre la forma de qualsevol tipus de dades. A l'exemple següent, la propietat *e1* permet emmagatzemar un objecte de qualsevol classe:

```
public class Prova<T>
{
  private T e1;
  public T E1
  {
    get { return e1; }
    set { e1 = value; }
  }
}
```

Els mètodes genèrics són similars, ja que permeten crear fragments de codi independents del tipus de dades amb què treballen (molt útil, per exemple, per a algorismes de cerca o ordenació). Tot seguit, mostrem un exemple de mètode genèric:

```
public static T[] OrdenacioBombolla <T> (T[] valors)
{
  ...
}
```

2.4. Gestió d'excepcions

Les excepcions són situacions imprevistes que poden esdevenir-se quan un programa està en execució. En C# podem gestionar les excepcions perquè el programa no acabi bruscament en cas d'error, i intentar solucionar-ho de la millor manera possible. Les excepcions són objectes de la classe *Exception* (o classes derivades d'aquesta), i quan es produeix una excepció podem capturar l'objecte i tractar-lo com s'escaigui.

Per a realitzar la gestió d'excepcions, disposem de *try*/*catch*/*finally*:

- ***try***: inclou el codi que pot generar alguna excepció.

- ***catch***: permet capturar l'objecte de l'excepció. Atès que es poden generar excepcions de diferents classes, podem tenir múltiples blocs *catch*.

- ***finally***: inclou codi que s'executa amb independència de si s'ha produït una excepció. És un bloc opcional, que se sol utilitzar si és necessari alliberar algun recurs, com ara una connexió a una base de dades o tancar un canal de lectura o escriptura.

En l'exemple de codi següent, intentem accedir fora dels límits d'un *array*, i es produeix una excepció, que capturem:

```
int[] nombres = new int[2];
try
{
  nombres[0] = 10;
  nombres[1] = 15;
  nombres[2] = 33;
  Console.WriteLine(nombres[3]);
}
catch (IndexOutOfRangeException)
{
  Console.WriteLine("Índex fora de rang");
}
catch (Exception ex)
{
  Console.WriteLine("Error desconegut: " + ex.Message);
}
finally
{
  Console.WriteLine("Aquí faríem la neteja.");
}
```

Quan es produeix una excepció, els blocs *catch* es revisen de manera seqüencial fins que se'n trobi un que gestioni el tipus d'excepció que s'ha produït. Si no se'n troba cap, l'excepció es propaga, és a dir, es cancel·la l'execució del mètode i es retorna el control del programa al mètode des del qual se n'havia fet la crida. Si aquest mètode tampoc no gestiona l'error, es torna a propagar, i així de manera recursiva fins que s'arriba al mètode principal. Si el mètode principal no gestiona l'error, el programa s'avorta i provoca una excepció de sistema. Una vegada gestionada una excepció en un bloc *catch*, continua l'execució del programa en la línia següent al bloc *try*/*catch* en el qual es va gestionar.

Cal tenir en compte que el primer bloc *catch* que coincideixi amb l'excepció produïda és el que la gestiona. Els altres *catch* no seran avaluats. Per això és important capturar tant les excepcions més específiques com les més genèriques, i en acabat, *Exception*, que és la primera en la jerarquia.

A tall d'exemple, si un *catch* gestiona l'excepció *IOException* i un altre gestiona *FileNotFoundException*, haurem de col·locar el segon abans que el primer perquè s'executi si es produeix una excepció de tipus *FileNotFoundException*, ja que una excepció d'aquest tipus és, alhora, del tipus *IOException* per herència. En cas que tinguem un bloc *catch* sense paràmetres, l'haurem de col·locar en últim lloc, perquè s'executa independentment de l'excepció que es produeixi.

2.4.1. Excepcions definides per l'usuari

El .NET Framework defineix una jerarquia de classes d'excepció que parteixen de la classe *Exception* com a arrel. Totes les classes que hereten d'*Exception* són tractades com a excepcions (i per tant es poden utilitzar en un *try*/*catch*).

Per a crear una excepció personalitzada, hem de crear una classe derivada d'*ApplicationException*, i sobreescriure els mètodes que corresponguin.

Finalment, ens falta saber com provocar una excepció. Això és útil quan el programa es troba una situació que li impedeix continuar, com per exemple quan l'usuari no ha indicat algun valor en un formulari. En aquest cas, podem crear una excepció especial, anomenada per exemple *FieldEmptyException*, i llançar-la quan ens falti algun valor. La interfície gràfica capturarà aquest error i avisarà l'usuari que ha d'omplir aquesta dada.

Per a llançar una excepció, utilitzem la instrucció *throw*, i creem un objecte del tipus de l'excepció que vulguem provocar, per exemple:

```
throw new FieldEmptyException ("dni");
```

Com es pot veure en l'exemple anterior, els objectes excepció poden tenir paràmetres; en aquest cas, passem el nom del camp que és buit. Depenent de la mida de l'aplicació, podem decidir si implementem excepcions personalitzades o no. Si l'aplicació és petita, podem fer servir directament la classe *Exception* o *ApplicationException*:

```
throw new Exception("El camp Nom no pot estar buit");
throw new ApplicationException("El camp Nom no pot estar buit");
```

3. .NET Framework

En aquest apartat us oferim una breu introducció a algunes de les classes bàsiques del .NET Framework, les classes de col·lecció i les d'entrada/sortida.

3.1. Classes bàsiques

Les classes del *namespaceSystem* ofereixen funcionalitats bàsiques. En aquest subapartat en veurem algunes de les més importants.

3.1.1. *System.Object*

La jerarquia de classes de .NET comença en la classe *Object*, és a dir, totes les classes i altres elements (interfícies, enumeracions, estructures, etc.) són, per herència directa o indirecta, subclasses d'*Object*. Per tant, totes les classes o elements hereten i poden sobreescriure els mètodes de la classe *Object* per a adequar-los a les necessitats que tenen.

Alguns dels seus mètodes són:

- ***Equals***: compara dos objectes i retorna un booleà que indica si són iguals o no ho són.
- ***GetHashCode***: retorna un número de *hash* que es fa servir per a emmagatzemar l'objecte en taules de *hash* (per exemple, la col·lecció *Hashtable*). Idealment, el número ha de ser diferent per a instàncies que representen objectes diferents.
- ***GetType***: retorna el tipus de dada de la instància actual.
- ***ToString***: retorna una representació textual de la instància actual.

Taules de *hash*

Una taula de *hash* és una taula en la qual cada element està identificat per una clau. Els elements s'insereixen i es recuperen utilitzant la clau corresponent com a referència.

3.1.2. *System.Convert*

La classe *Convert* conté una sèrie de mètodes estàtics molt útils que permeten fer conversions entre diferents tipus de dades. Hi ha un mètode de conversió per a cada tipus de dada bàsic: *ToInt32*, *ToDouble*, *ToChar*, *ToString*, etc. Per exemple, podem convertir un *double* a *Int32* de la manera següent:

```
double d = 4.7;
int i = Convert.ToInt32(d);
```

3.1.3. *System.Math*

Conté mètodes estàtics per a realitzar operacions matemàtiques com ara:

Operacions de la classe *System.Math*	
Abs	Retorna el valor absolut d'un nombre
Cos	Retorna el cosinus d'un angle
Exp	Retorna l'elevat a una potència
Log	Retorna el logaritme d'un nombre
Pow	Retorna la potència d'un nombre
Round	Retorna un nombre arrodonit
Sin	Retorna el sinus d'un angle
Sqrt	Retorna l'arrel quadrada d'un nombre
Tan	Retorna la tangent d'un angle

La classe *Math* també inclou les constants *E* i *PI*.

3.1.4. *System.Random*

La classe *Random* permet generar nombres aleatoris. En realitat, els nombres generats simulen aleatorietat a partir d'un nombre inicial anomenat llavor[7]. El constructor de la classe permet especificar un *seed* concret:

(7) De l'anglès, *seed*.

```
Random r = new Random(45);
```

Si sempre escollim el mateix *seed*, obtindrem la mateixa seqüència de nombres aleatoris. Per a augmentar l'aleatorietat, el constructor per defecte de la classe escull un *seed* relacionat amb l'hora del processador.

Una vegada creada una instància de la classe *Random*, podem obtenir nombres aleatoris utilitzant el mètode *Next*. Per exemple, la instrucció següent retorna un enter entre el 0 i el 10:

```
int i = r.Next(0, 10);
```

3.1.5. *System.String*

String és una classe que ens permet treballar amb cadenes de caràcters. *String* és un tipus especial, ja que es comporta com un tipus valor (no cal fer servir la paraula clau *new* per a definir una variable de tipus cadena), encara que en realitat és de tipus referència. Aquest tipus es pot escriure indistintament com *string* o *String*.

Si davant de la cadena de caràcters posem el caràcter @, podem evitar els caràcters d'escapada i escriure caràcters com una contrabarra (\) o un salt de línia. Resulta molt útil per a escriure rutes de directoris:

```
string s1 = "Hola això és un string"
string s2 = @"c:\test\prueba.cs"
```

La classe *String* té molts mètodes útils:

Mètodes de la classe *System.String*	
CompareTo	Compara dues cadenes alfanumèriques.
IndexOf	Retorna la posició d'una subcadena.
Replace	Reemplaça una subcadena per una altra.
Substring	Retorna una subcadena determinada.
ToLower	Retorna la mateixa cadena passada a minúscules.
ToUpper	Retorna la mateixa cadena passada a majúscules.
Trim	Elimina els espais a l'inici i al final de la cadena.

3.1.6. *System.DateTime*

DateTime permet emmagatzemar una data i una hora. Podem accedir a aquestes dades mitjançant propietats com *Year*, *Month*, *Day*, *Hour*, *Minute*, *Second*, i podem obtenir la data/hora actual mitjançant la propietat *Now*.

D'altra banda, conté mètodes per a afegir unitats de temps al valor actual (*AddDays*, *AddMoths*, etc.), i també diversos mètodes que permeten convertir un *DateTime* en altres tipus i viceversa. Per exemple, el mètode *Parse* permet convertir un *string* amb una data, a un tipus *DateTime*.

L'exemple següent ens indica si l'any actual és de traspàs o no:

```
DateTime now = DateTime.Now;
int year = now.Year;
if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
  Console.WriteLine("Enguany és any de traspàs");
```

3.1.7. *System.Array*

La classe *Array* conté una sèrie de propietats que hereten tots els *arrays*, entre les quals podem esmentar:

- ***Length***: retorna el nombre total de posicions de l'*array*.

- ***Rank***: retorna el nombre de dimensions de l'*array*.

- ***Clear***: inicialitza les posicions indicades de l'*array*.

- ***Copy***: permet copiar parts d'un *array* a un altre.

- ***IndexOf***: cerca un element en l'*array*, i en retorna la posició.

- ***Sort***: ordena els elements d'un *array* unidimensional. Per a poder-los ordenar, és necessari que els elements de l'*array* implementin la interfície *IComparable* o proporcionar una instància d'una classe que implementi la interfície *IComparer* i permeti comparar dos elements del tipus de l'*array* entre ells.

3.1.8. *System.Environment*

La classe *Environment* permet consultar diferents característiques de l'entorn en el qual s'executa l'aplicació. En destaquem els següents:

- ***CurrentDirectory***: obté la ruta d'accés completa del directori en el qual s'ha iniciat l'aplicació.
- ***MachineName***: obté el nom de l'equip en el qual s'està executant l'aplicació.
- ***GetEnvironmentVariable***: permet consultar variables d'entorn. Vegem-ne un exemple per consultar el valor de la variable *PATH*:

```
string path = Environment.GetEnvironmentVariable("PATH");
```

3.2. Col·leccions de dades

Els *arrays* permeten emmagatzemar eficientment objectes, i accedir-hi per la posició en la qual estan. No obstant això, els *arrays* tenen la limitació que tots els seus elements han de ser del mateix tipus, i a més, cal indicar la longitud exacta de l'*array* en el moment de crear-lo.

Per tal de suplir les limitacions dels *arrays*, en el *namespace System.Collections* disposem d'un conjunt de classes de col·lecció, mitjançant les quals podrem fer servir llistes enllaçades, piles, cues, taules *hash*, etc. Alguns d'aquests tipus són:

- ***ArrayList***: permet emmagatzemar objectes, i hi podem accedir per la seva posició dins de l'estructura. És similar a un *array*, però amb la diferència que *ArrayList* és una estructura dinàmica, que sol·licita o allibera memòria segons li calgui.

- ***Queue***: representa una cua (estructura de dades de tipus *FIFO*).
- ***Stack***: representa una pila (estructura de dades de tipus *LIFO*).
- ***Hashtable***: representa una taula *hash* o diccionari, en la qual els elements s'identifiquen amb una clau.

També podem utilitzar col·leccions genèriques on, en crear-les, hem d'indicar el tipus de dades que emmagatzemaran. La taula següent mostra algunes de les col·leccions genèriques i les seves classes de col·lecció equivalents:

Col·lecció genèrica	Col·lecció equivalent
List<T>	ArrayList
Dictionary<K,V>	Hastable
Queue<T>	Queue
Stack<T>	Stack

3.3. Entrada/Sortida

El *namespace System.IO* conté tota la funcionalitat d'entrada/sortida. L'entrada/sortida té lloc mitjançant *streams*, que són fluxos de lectura o escriptura sobre un cert mitjà d'emmagatzemament com, per exemple, la memòria, el teclat, la pantalla, o els fitxers de disc.

Streams

Gràcies a l'abstracció que ofereixen els *streams*, podem accedir a diferents tipus de dispositius d'entrada/sortida fent ús d'unes classes molt similars.

Els fluxos de dades estan representats per la classe *Stream*, que permet dur a terme diferents operacions:

Operacions de la classe *Stream*	
length	Retorna la longitud total del flux.
position	Retorna la posició actual en el flux.
close	Tanca el flux.
read	Llegeix una seqüència de bytes del flux.
seek	Canvia la posició actual en el flux.
write	Escriu una seqüència de bytes en el flux.

La classe *Stream* és abstracta; per tant, no es pot instanciar directament, però hi ha diverses subclasses que sí que es poden utilitzar:

Subclasses de la classe *Stream*	
FileStream	Flux associat a un fitxer

Subclasses de la classe *Stream*	
MemoryStream	Flux en memòria
CryptoStream	Flux de dades encriptades
NetworkStream	Flux associat a una connexió de xarxa
GZipStream	Flux de dades comprimides

Per a llegir/escriure tipus de dades en un *stream*, se sol utilitzar un lector o escriptor[8], com ara:

[(8)]De l'anglès, *reader* i *writer*.

Lectors i escriptors d'un *stream*	
StreamReader	Permet llegir caràcters.
StreamWriter	Permet escriure caràcters.
StringReader	Permet llegir cadenes de caràcters.
StringWriter	Permet escriure cadenes de caràcters.
BinaryReader	Permet llegir tipus de dades primitives.
BinaryWriter	Permet escriure tipus de dades primitives.

Generalment, per llegir i escriure fitxers s'utilitzen les classes *StreamReader* i *StreamWriter*, que es poden inicialitzar directament especificant el nom del fitxer que es vol obrir, sense necessitat de crear primer un objecte de tipus *FileStream*. A més, aquestes dues classes contenen els mètodes *ReadLine* i *WriteLine*, que permeten llegir o escriure respectivament línies senceres del fitxer. En l'exemple següent, veiem un mètode que llegeix un fitxer i el mostra per pantalla:

```
public void LlegirFitxer (string file)
{
  StreamReader sr = new StreamReader (file);
  string s = sr.ReadLine ();
  while (s!=null)
  {
    Console.WriteLine (s);
    s = sr.ReadLine ();
  }
  sr.Close();
}
```

Al final d'una operació amb *streams* és important tancar l'objecte lector o escriptor, de manera que també es tanqui el flux de dades. En cas de no fer-ho, deixaríem el recurs bloquejat i la resta d'aplicacions no podrien accedir-hi.

4. ADO.NET

Aquest apartat tracta principalment sobre ADO.NET des del punt de vista de la seva arquitectura i les classes principals que el componen, i tot seguit s'hi presenta una introducció a LINQ.

4.1. Una introducció a ADO.NET

ADO.NET és l'API d'accés a fonts de dades de .NET. Però abans d'entrar en detall, fem un breu repàs de les tecnologies d'accés a dades que hi ha hagut en l'entorn de Microsoft.

Inicialment, les primeres biblioteques de desenvolupament per a l'accés a dades eren específiques per a cada tipus de base de dades concreta, per la qual cosa no eren interoperables entre elles. El 1989, diverses empreses de programari (Oracle, Informix, Ingres, DEC i d'altres) van formar l'SQL Access Group amb l'objectiu de definir i promoure estàndards per a la interoperabilitat entre bases de dades, i van publicar l'especificació SQL CLI, en què es defineixen les interfícies que s'han d'utilitzar per a executar sentències SQL des d'altres llenguatges de programació.

El 1992, Microsoft va llançar Open Database Connectivity 1.0 (**ODBC**), basat en SQL CLI. El problema és que ODBC era complex, lent i no es basava en el model de components COM.

Microsoft va desenvolupar diverses tecnologies sobre ODBC:

- **Data Access Object** (**DAO**) va ser desenvolupada per a accedir a Microsoft Access. DAO era molt eficient per a bases de dades locals, però no estava preparada per a aplicacions amb múltiples usuaris simultanis.
- **Remote Data Objects** (**RDO**) va aparèixer posteriorment. El seu avantatge és que permet consultes complexes i accessos simultanis.

Atenció!

No s'ha de confondre l'API DAO de Microsoft amb el patró de disseny Data Access Object.

El 1996 va aparèixer la nova tecnologia Object Linking and Embedding DataBase (**OLEDB**), que és la successora d'ODBC. Aquesta tecnologia té múltiples avantatges, com ara un rendiment molt millor i que està basada en COM. Per damunt d'OLE DB Microsoft va desenvolupar **ActiveX Data Objects** (**ADO**), amb l'objectiu de substituir DAO i RDO.

El 2002 va aparèixer **ADO.NET** com a tecnologia d'accés a fonts de dades de .NET, que hereta les millors característiques d'ADO, i proporciona noves funcionalitats com, per exemple, la possibilitat de treballar tant de manera connectada com desconnectada de les bases de dades.

Les **característiques principals d'ADO.NET** són:

a) Permet treballar tant de manera connectada com desconnectada de l'origen de dades. Un dels avantatges de l'accés desconnectat és que s'aconsegueix una major escalabilitat, atès que les connexions dels usuaris no es mantenen per períodes llargs, amb la qual cosa es pot permetre l'accés de més usuaris.

b) Té una forta integració amb l'XML, de manera que se'n facilita tant la compartició i interpretació com treballar-hi.

c) És independent del llenguatge de programació que es fa servir.

d) No solament permet accedir a bases de dades, sinó també a altres fonts de dades com ara fulls de càlcul, XML, text, etc.

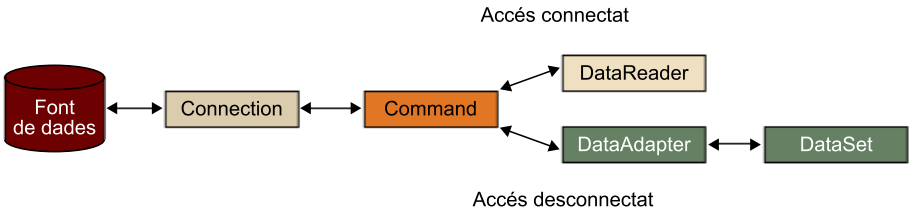
Com hem comentat, ADO.NET permet treballar de dues maneres:

- **Accés connectat**: necessita una connexió permanent amb la base de dades (això era el més habitual fins a l'arribada d'ADO.NET).

- **Accés desconnectat**: un subconjunt de les dades de la font de dades es copien en un *dataset* i, si després es produeixen canvis en el *dataset*, aquests es propaguen a la font de dades.

La figura següent mostra les classes de .NET que intervenen en cada cas:

Figura 2. Modes d'accés de ADO.NET



Les classes principals d'ADO.NET són les següents:

- ***Connection***: fa la connexió amb una font de dades.

- ***Command***: permet fer consultes o operacions de modificació contra una font de dades.

- ***DataReader***: permet accedir als resultats d'una consulta realitzada contra una font de dades.

- ***DataAdapter***: llegeix les dades i les carrega al *dataset*. Si s'hi produeixen canvis, el *data adapter* serà l'encarregat de sincronitzar tots els canvis en la font de dades.

- ***DataSet***: permet emmagatzemar i manipular dades en memòria.

Per establir una **connexió amb la font de dades**, en primer lloc hem de seleccionar el proveïdor de dades adequat, i després cal tenir en compte les cadenes de connexió.

4.1.1. Proveïdors de dades

Un proveïdor de dades consta d'un conjunt de classes que permeten l'accés i la comunicació amb les fonts de dades. Dins del .NET Framework, s'inclouen els proveïdors de dades següents:

Altres proveïdors

Hi ha molts altres proveïdors de dades que no vénen integrats amb el .NET Framework, i n'hi ha tant de gratuïts com de comercials.

- **Proveïdor de dades per a ODBC**: proveïdor genèric que permet accedir a qualsevol font de dades mitjançant el driver ODBC corresponent (s'ha de tenir instal·lat prèviament).

- **Proveïdor de dades per a OLEDB**: proveïdor genèric que permet accedir a qualsevol font de dades mitjançant el driver OLE DB (s'ha de tenir instal·lat prèviament). Si el comparem amb el proveïdor ODBC, aquest és més recomanable ja que, entre altres motius, és més ràpid.

- **Proveïdor de dades per a SQL Server**: proveïdor específic per a SQL Server 7.0 o posterior. La comunicació amb SQL Server té lloc sense capes intermèdies (ni OLEDB ni ODBC), amb la qual cosa el rendiment és encara millor.

- **Proveïdor de dades per a Oracle**: és un proveïdor de dades específic per a Oracle 8.1.7 o posterior, que també ofereix el millor rendiment possible, en no fer ús de cap capa intermèdia (ni OLE DB ni ODBC).

Tots els proveïdors de dades han d'oferir les classes següents:

Classe	Descripció
xxxConnection	Permet establir una connexió a un tipus de font de dades.
xxxCommand	Permet executar una ordre SQL en una font de dades.
xxxDataReader	Permet llegir un conjunt de dades de la font de dades
xxxDataAdapter	Permet carregar i actualitzar dades en un objecte *DataSet*

4.1.2. Cadenes de connexió

La cadena de connexió és una cadena de caràcters que identifica les característiques de la connexió amb una font de dades determinada. La cadena de connexió inclou la localització del servidor, el nom de la base de dades, l'usuari, la contrasenya, o altres opcions específiques del proveïdor de dades utilitzat. La taula següent mostra els paràmetres més comuns:

Paràmetre	Descripció
Data Source	Nom del proveïdor de la connexió
Initial Catalog	Nom de la base de dades
User ID	Usuari d'accés a la font de dades
Contrasenya	Contrasenya d'accés a la font de dades

A tall d'exemple, el codi següent es connecta a una base de dades SQL Server amb l'usuari "sa", contrasenya xxx, i accedeix a la base de dades Northwind del servidor miBBDD:

```
SqlConnection myConnection = new SqlConnection();
myConnection.ConnectionString = "Data Source=miBBDD;" +
                                "Initial Catalog=Northwind;" +
                                "User ID=sa;Password=xxx"
myConnection.Open();
```

Accés connectat

L'accés connectat consisteix a obrir una connexió amb la font de dades, executar una sèrie de sentències SQL i tancar la connexió. La classe *Command* ofereix els mètodes següents per executar les sentències:

Mètode	Descripció
ExecuteScalar	Executa una sentència que retorna un únic valor.
ExecuteReader	Executa una sentència que retorna un conjunt de files.
ExecuteNonQuery	Executa una sentència que modifica l'estructura o les dades de la base de dades. Torna el nombre de files afectades.
ExecuteXmlReader	Executa una sentència que retorna un resultat XML.

Sentències que no retornen valors

Utilitzarem el mètode *ExecuteNonQuery* per a executar sentències SQL que no retornen valors. L'exemple següent crea una taula anomenada *A*:

```
SqlCommand cmd = new SqlCommand (
```

```
  "CREATE TABLE A (A INT, PRIMARY KEY (A))", conn);
cmd.ExecuteNonQuery ();
```

En el cas de les sentències SQL de manipulació (DML), el mètode *ExecuteNonQuery* retorna el nombre de files afectades. L'exemple següent modifica el preu dels productes i obté el nombre de files afectades:

```
SqlCommand cmd = new SqlCommand (
  "UPDATE PRODUCTES SET preu = preu + 10", conn);
int nomFiles = cmd.ExecuteNonQuery ();
```

Sentències que retornen un únic valor

Utilitzarem el mètode *ExecuteScalar* quan la sentència retorni un únic valor. L'exemple següent torna el nombre de files d'una taula:

```
SqlCommand cmd = new SqlCommand (
  "SELECT COUNT(*) FROM PRODUCTES", conn);
int nomFiles = (int)(cmd.ExecuteScalar ());
```

Sentències que retornen un conjunt de valors

En general, l'execució d'una sentència *SELECT* o un procediment emmagatzemat retorna un conjunt de dades. Per a executar aquest tipus de sentències utilitzem el mètode *ExecuteReader* de la classe *Command*, que retorna un objecte *DataReader* que ens permet consultar les files obtingudes.

Un *DataReader* és un cursor que permet iterar cap endavant sobre un conjunt de files (no es poden recuperar elements anteriors), com a resultat de l'execució d'una sentència SQL o procediment emmagatzemat. Per a poder utilitzar un *DataReader* s'ha de mantenir oberta la connexió amb la base de dades; no podem tancar-la i després recórrer al *DataReader*. Per això, es denomina *accés a dades connectat*.

En l'exemple següent, fem una consulta i obtenim els resultats següents:

```
SqlCommand cmd = new SqlCommand(
  "SELECT id, descr FROM usuaris", conn);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
  int id = reader.GetInt32(0);
  String descr = reader.GetString(1);
}
reader.Close();
conn.Close();
```

Comprovació de valors *null*

Podem comprovar quan el valor d'una columna és nul amb el mètode *IsDbNull* del *DataReader*:

```
if (!reader.IsDBNull (3)) // Si la tercera columna no és null
{
  int i = dr.GetInt32(3);
}
```

A partir de la versió 2.0 del .NET Framework, gràcies als tipus anul·lables, podem executar el mètode *GetXXX* sense comprovacions addicionals:

```
int? i = dr.GetInt32(3);
```

Tipus anul·lables

Els tipus anul·lables són una extensió dels tipus valor de .NET, i permeten que el tipus de dades pugui contenir també el valor *null*. Per indicar que un tipus és anul·lable, afegim un interrogant de tancament (?) al costat del tipus de dades.

Procediments emmagatzemats

Els procediments emmagatzemats poden retornar resultats o no retornar-ne. Si retornen resultats, els executarem mitjançant el mètode *ExecuteReader* de la classe *Command*, mentre que si no en retornen, els executarem mitjançant el mètode *ExecuteNonQuery*.

Per a executar un procediment emmagatzemat cal crear un *SqlCommand*, al qual s'indica el nom i els paràmetres del procediment:

```
SqlCommand cmd = new SqlCommand ("ActualitzarPreu", conn);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add ("@producteID", 4);
cmd.Parameters.Add ("@nouPreu", 50);

int modified = cmd.ExecuteNonQuery ();
Console.WriteLine ("S'han modificat " + modified + " files");
```

Transaccions

Transaccions

Una transacció és un conjunt de sentències relacionades que, o bé s'executen totes, o no se'n pot executar cap. Es diu que una transacció fa *commit* o *rollback*, respectivament.

ADO.NET permet gestionar transaccions de manera senzilla.

En l'exemple següent, executem dues sentències SQL dins d'una transacció:

```
String connString = "Data Source=miBBDD;" +
  "Initial Catalog=Northwind;" +
  "User ID=sa;Password=xxx";
SqlConnection conn = new SqlConnection(connString);
```

```
conn.Open();
SqlTransaction tran = conn.BeginTransaction();
SqlCommand command1 = new SqlCommand(
  "DELETE FROM User WHERE Id = 100", conn, tran);
SqlCommand command2 = new SqlCommand(
  "DELETE FROM User WHERE Id = 200", conn, tran);

try
{
  command1.ExecuteNonQuery();
  command2.ExecuteNonQuery();
  tran.Commit();
}
catch (SqlException)
{
  tran.Rollback();
}
finally
{
  conn.Close();
}
```

Accés desconnectat

En aquest subapartat expliquem els diversos tipus d'accés desconnectat que podem trobar.

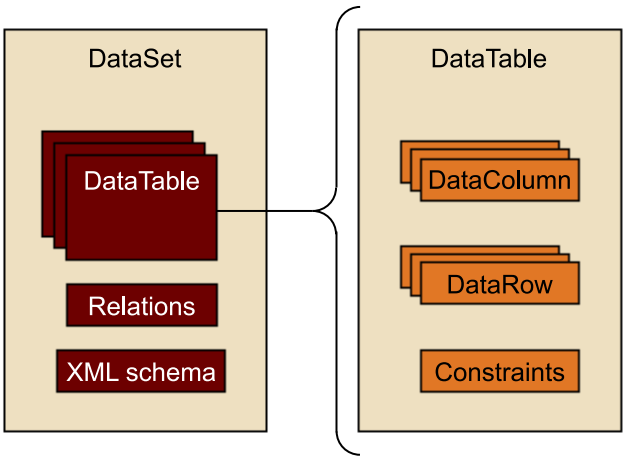
Datasets no tipats

Un *DataSet* és una còpia en memòria d'una part d'una base de dades. Constitueix una vista "desconnectada" de les dades, és a dir, existeix en memòria sense una connexió activa a una base de dades que contingui la corresponent taula o vista. L'objecte *DataSet* desa les dades i la seva estructura en XML, la qual cosa permet, per exemple, enviar o rebre un arxiu XML per mitjà d'HTTP.

Un *DataSet* està format per taules (*DataTables*) i una llista de relacions entre aquestes (*Relations*). A més, el *DataSet* manté un *XML schema* amb l'estructura de les dades. Un *DataTable*, al seu torn, està integrat per columnes (*DataColumn*) i conté una sèrie de files de dades (*DataRow*). També pot tenir definits una sèrie de restriccions (*Constraints*), com *Primary key*, *Foreign key*, *Unique* o *Not null*.

Les files constitueixen les dades que s'emmagatzemen en el *DataTable*. De cada *DataRow*, se'n guarden dues còpies, una de la versió inicial (la que es recupera de la base de dades), i una altra de la versió actual, per a identificar els canvis introduïts en el *DataSet*.

Figura 3. Estructura d'un *DataSet*



L'exemple següent crea un objecte *DataTable* que representa la taula PRODUCTES d'una font de dades, i l'afegeix al *DataSet* MAGATZEM:

```
DataSet ds = new DataSet("MAGATZEM");
DataTable dt = ds.Tables.Add("PRODUCTES");
dt.Columns.Add("id", typeof (Int32));
dt.Columns.Add("nom", typeof (String));
dt.Columns.Add("preu", typeof (Double));
```

Per tal derecuperar taules o columnes de les col·leccions *Tables* i *Columns*, podem accedir per posició o pel nom de l'element. A l'exemple següent, recuperem la columna "id" de la taula PRODUCTES:

```
// accés per posició
DataColumn dc = ds.Tables[0].Columns[0];

// accés per nom
DataColumn dc = ds.Tables["PRODUCTES"].Columns["id"];
```

Datasets tipats

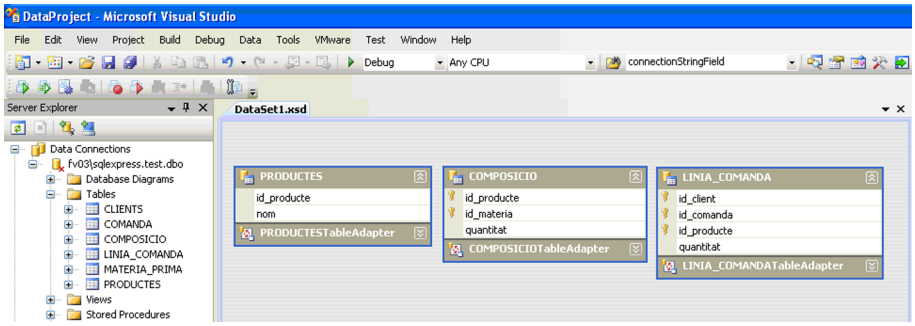
Un *dataset* tipat és una classe derivada de *DataSet*, i que té un esquema de taules i relacions predefinit. A més, proporciona mètodes i propietats per a accedir a les taules, files i columnes del *DataSet*, utilitzant-ne els noms i permetent-hi un codi molt més senzill i llegible.

El .NET Framework SDK proporciona l'eina xsd.exe que, a partir d'un *schema* XML (XSD) que defineix l'estructura del *DataSet*, crea la classe *DataSet* que correspon a l'estructura indicada. Podem compilar aquesta classe per separat, o afegir-la directament amb la resta de classes de l'aplicació.

Per a facilitar la tasca de creació de *datasets* amb tipus, Visual Studio proporciona eines de disseny visual a aquest efecte. Per a això, cal afegir un nou element al projecte de tipus *DataSet*, cosa que dispara el dissenyador de *datasets*, on podem definir-ne l'estructura en forma d'XSD.

Una altra opció més ràpida és crear l'estructura d'un *DataSet* a partir d'una base de dades existent. Per a això, a la finestra de l'explorador de servidors, obrim la llista de taules de la base de dades i arrosseguem les taules que vulguem afegir al *DataSet* directament al dissenyador de *DataSet*.

Figura 4. Creació d'un *DataSet* tipat



Una vegada creada l'estructura d'un *dataset*, podem afegir-hi noves files de dades, fer modificacions o eliminacions. Una vegada fets aquests canvis, podem acceptar-los o rebutjar-los amb els mètodes *AcceptChanges* i *RejectChanges*. Concretament, si executem *AcceptChanges* sobre una fila, tots els canvis passen a ser dades permanents. En canvi, si executem *RejectChanges*, rebutjarem els canvis i la fila torna a l'estat anterior a tots aquests canvis.

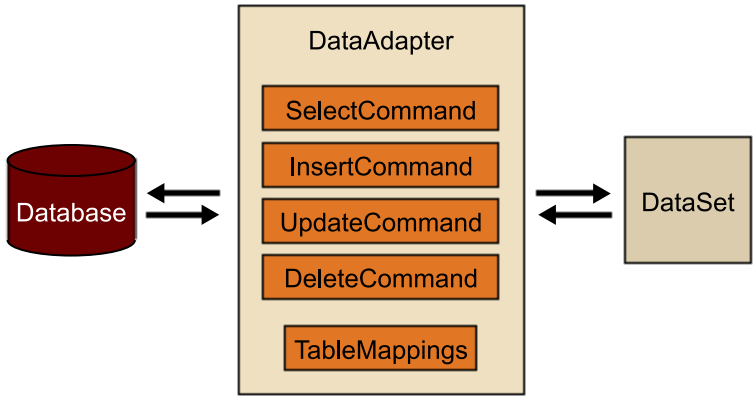
La classe *DataAdapter*

La classe *DataAdapter* actua d'intermediària entre la font de dades i un *DataSet*. El *DataAdapter* és l'encarregat de connectar-se a la font de dades, executar la consulta SQL que volem fer, i emplenar el *DataSet* amb les dades obtingudes. També és l'encarregat d'actualitzar la font de dades amb els canvis realitzats en el *DataSet*, una vegada s'ha acabat de treballar-hi.

Aquesta classe forma part del proveïdor de dades que es fa servir; per exemple, per al proveïdor SQLServer.NET, serà *SqlDataAdapter*, i per al d'OLEDB, *OleDbDataAdapter*.

Com veiem en el diagrama següent, un *DataAdapter* conté una sèrie d'objectes *Command* que representen les sentències de consulta (*SelectCommand*), inserció de noves dades (*InsertCommand*), modificació de dades (*UpdateCommand*), i eliminació de dades (*DeleteCommand*) respectivament. A més, conté una propietat *TableMappings* que relaciona les taules del *DataSet* amb les taules homòlogues de la font de dades.

Figura 5. La classe *DataAdapter*



La classe *DataAdapter* proporciona el mètode *Fill*, que permet emplenar un objecte *DataSet* amb els resultats de la consulta SQL representada per l'objecte *SelectCommand*. L'exemple següent mostra com crear un objecte *DataAdapter*, i com emplenar un *dataset* amb el mètode *Fill*:

```
SqlConnection conn = new SqlConnection(strConn);

SqlDataAdapter da = new SqlDataAdapter (
  "SELECT id, nom FROM CLIENTS", conn);

DataSet ds = new DataSet ();

da.Fill(ds, "CLIENTS");
```

Com es mostra en l'exemple, no cal de cridar explícitament els mètodes *Open* i *Close* de la connexió. El mètode *Fill* comprova si la connexió està oberta o no ho està, i si no ho està, l'obre i la tanca en acabar.

Una vegada s'han fet tots els canvis pertinents, utilitzem de nou la classe *DataAdapter* per a actualitzar les dades de la font de dades amb les modificacions fetes al *DataSet* amb el mètode *Update*, que fa servir les sentències SQL representades per les propietats *InsertCommand*, *UpdateCommand* i *DeleteCommand*, per tal d'inserir les files noves, modificar les ja existents, i esborrar les eliminades, respectivament.

- **Problemes de concurrència**
  L'ús de *DataSets* en una base de dades a la qual accedeixen múltiples usuaris afegeix el problema de la gestió de la concurrència. Mentre un usuari està treballant amb les dades en una còpia local, en la font de dades real s'hi pot estar produint una actualització, la qual cosa podria invalidar les dades que havíem llegit, quan encara hi estàvem treballant. En general, hi ha diverses tècniques de gestió de la concurrència:

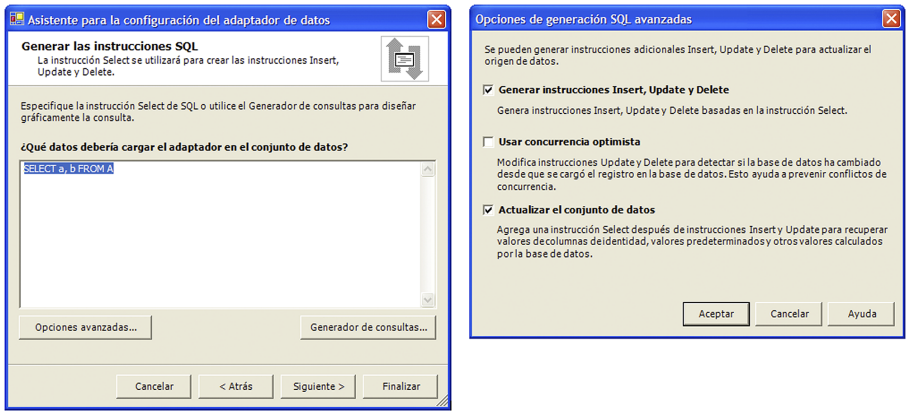
– **Concurrència pessimista**: quan una fila és llegida, aquesta queda bloquejada sigui qui sigui qui la vulgui llegir, fins que aquell qui la posseeix l'alliberi.

– **Concurrència optimista**: les files estan sempre disponibles per a la seva lectura i poden ser llegides per diferents usuaris alhora. Quan algú intenta modificar una fila que havia estat modificada, es produeix un error i no es pot fer la modificació.

– ***Last win***: aquesta tècnica implica que no hi ha cap control; simplement, l'última actualització és la bona.

Visual Studio només permet escollir entre concurrència optimista o *last win* en crear un objecte *DataAdapter* amb l'assistent de configuració (vegeu la figura següent). En cas de no seleccionar concurrència optimista, s'aplica per defecte concurrència de tipus *last win*.

Figura 6. Assistent del *DataAdapter*



La classe *DataView*

Un objecte *DataView* és similar a una vista d'una base de dades, on podem fer filtrages o ordenacions de les dades. A l'exemple següent, creem una vista ordenada per nom, i amb un filtre de preu:

```
DataView dv = new DataView (ds.Tables("PRODUCTES"));
dv.Sort = "nom";
dv.RowFilter = "preu > 100";
```

4.2. LINQ

LINQ afegeix capacitats de consulta i modificació de dades als llenguatges C# i Visual Basic .NET. LINQ no permet només treballar amb bases de dades, sinó que està dissenyat per treballar amb qualsevol tipus d'origen de dades, com ara llistes d'objectes, arxius XML, etc.

La diferència principal entre fer servir ADO.NET o LINQ per a consultar una font de dades és que amb LINQ les consultes es comproven de manera sintàctica en temps de compilació.

Disposem de les implementacions de LINQ següents:

1) LINQ to Objects: permet fer consultes sobre col·leccions d'objectes. És el que farem servir en tots els exemples d'aquest subapartat.

2) LINQ to XML: permet fer consultes sobre dades en format XML.

3) LINQ to ADO.NET: permet consultar bases de dades a través d'ADO.NET. N'hi ha tres variants:

a) LINQ to SQL: permet fer consultes sobre bases de dades relacionals, tot i que inicialment només amb SQL Server.

b) LINQ to DataSet: permet fer consultes sobre DataSets.

c) LINQ to Entities: permet fer consultes sobre un model conceptual de dades. Està relacionat amb ADO.NET Entity Framework, que permet treballar contra un model conceptual de dades, sense preocupar-se de l'estructura real de la base de dades.

4.2.1. Sintaxi de LINQ

La sintaxi de LINQ és similar al llenguatge SQL. En LINQ, el resultat d'una consulta és sempre un tipus enumerable (concretament, és una instància d'*IEnumerable<T>*), amb la qual cosa sempre podrem recórrer el resultat tal com faríem amb una enumeració qualsevol.

Atenció!

Per a treballar amb LINQ s'ha d'afegir la sentència "using System.Linq", i el projecte ha d'estar configurat per a treballar amb .NET Framework 3.5.

L'exemple següent fa una consulta que retorna els noms de les persones de l'*array* el lloc de naixement de les quals sigui Barcelona, que són Rosa i Juan:

```
Persona[] persones = new Persona[] {
  new Persona ("Juan", "Barcelona", 20),
  new Persona ("Pedro", "Londres", 30),
  new Persona ("Maria", "Lisboa", 40),
  new Persona ("Rosa", "Barcelona", 25)};

var persones_BCN =
  from p in persones
  where (p.Ciutat == "Barcelona")
  orderby p.Edat descending, p.Nom
  select (p);

foreach (Persona p in persones_BCN)
```

```
    Console.WriteLine(p.Nom);
```

La paraula clau *var* s'utilitza per a inferir automàticament el tipus de dada del resultat. En l'exemple anterior, seria equivalent haver escrit *IEnumerable<Persona>* en lloc de *var*. Alguns dels operadors de LINQ són:

- ***from***: indica la font d'informació sobre la qual s'executarà la consulta.
- ***where***: permet especificar les restriccions o filtres a aplicar les dades.
- ***select***: indica els camps a retornar en el resultat.
- ***orderby***: permet ordenar els resultats per un o més criteris d'ordenació, i podem indicar-ne l'ordre com *ascending* (per defecte) o *descending*.
- ***group by***: permet agrupar els resultats segons un criteri determinat. L'exemple següent mostra com agrupar les persones per ciutat:

```
Persona[] persones = new Persona[] {
  new Persona ("Juan", "Barcelona", 20),
  new Persona ("Pedro", "Londres", 30),
  new Persona ("Maria", "Lisboa", 40),
  new Persona ("Rosa", "Barcelona", 25)};

var grups = from p in persones
    group p by p.Ciutat;

foreach (var grup in grups)
{
  Console.WriteLine("Ciutat: " + grup.Key);
  Console.WriteLine("Cont: " + grup.Count());
  foreach (Persona persona in grup)
  {
    Console.WriteLine(" " + persona.Nom);
  }
}
```

En executar l'exemple anterior, obtenim el resultat següent:

```
Ciutat: Barcelona
Cont: 2
  Juan
  Rosa
Ciutat: Londres
Cont: 1
  Pedro
Ciutat: Lisboa
```

```
Cont: 1
  Maria
```

- ***join***: l'operador *join* permet definir una relació entre dues classes o entitats dins de la consulta. La seva semàntica és similar a la del *join* d'SQL, ja que el que fa és creuar les dues taules (o classes) en funció de l'expressió d'enllaç especificada. Tot seguit, mostrem un exemple en el qual fem servir la instrucció *join* per a combinar les persones amb les ciutats, i obtenir un llistat amb el nom de la persona, la seva ciutat, i el país:

```
Persona[] persones = new Persona[] {
  new Persona ("Juan", "Barcelona", 20),
  new Persona ("Pedro", "Londres", 30),
  new Persona ("Maria", "Lisboa", 40),
  new Persona ("Rosa", "Barcelona", 25)};

Ciutat[] ciutats = new Ciutat[] {
  new Ciutat ("Barcelona", "Espanya"),
  new Ciutat ("Londres", "Regne Unit"),
  new Ciutat ("Lisboa", "Portugal")};

var llistat =
  from p in persones
  join c in ciutats
  on p.Ciutat equals c.Nom
  select new {p.Nom, p.Ciutat, c.Pais};

foreach (var p in llistat)
  Console.WriteLine(p.Nom+"\t"+p.Ciutat+"\t"+p.Pais);
```

En executar l'exemple anterior, obtenim el resultat següent:

```
Juan       Barcelona        Espanya
Pedro      Londres          Regne Unit
Maria      Lisboa           Portugal
Rosa       Barcelona        Espanya
```

Agregats

Els agregats són mètodes que retornen el resultat de fer una determinada operació sobre els elements de la llista de valors sobre la qual s'aplica. Tot seguit, s'indica la utilitat de cada un d'ells:

Mètode	Descripció
Count	Retorna el nombre d'elements de la llista.

Mètode	Descripció
Min	Retorna l'element menor.
Max	Retorna l'element més gran.
Sum	Retorna la suma dels valors de la llista.
Average	Retorna una mitjana dels valors de la llista

L'exemple següent mostra l'edat mínima de les persones de l'*array*, la qual cosa, executada sobre l'*array* de persones de l'exemple anterior, ens donaria com a resultat el valor 20:

```
var edat_minima = ( from p in persones
                    select p.Edat).Min();
Console.WriteLine(edat_minima);
```

5. Windows Forms

En aquest apartat s'introdueix la tecnologia Windows Forms, que permet crear aplicacions d'escriptori.

Windows Forms és una tecnologia de Microsoft que permet desenvolupar aplicacions d'escriptori de manera senzilla. Abans d'aparèixer, es feien servir les MFC (Microsoft Foundation Classes), en què la programació era bastant més complexa.

Windows Forms apareix l'any 2002, i consta d'un conjunt de classes que permeten desenvolupar interfícies d'usuari, sia afegint controls gràfics que gestionem mitjançant esdeveniments, o realitzant crides a baix nivell mitjançant les classes de System.Drawing, amb les quals podem dibuixar, escriure text, processar imatges, etc.

L'ús per part del programador de biblioteques gràfiques més complexes, com la llibreria GDI+ de Win32, DirectX o OpenGL (Open Graphics Library), queda relegat només a les situacions que necessiten funcionalitats gràfiques més avançades, com processament d'imatges, animacions, 3D, etc.

Llibreria GDI+

Encara que el programador no utilitzi directament GDI+ (de l'anglès, *graphics device interface*), les crides a l'API de WinForms s'acaben traduint internament a crides a GDI+.

Un formulari és la finestra utilitzada per a presentar la informació, o rebre les entrades de l'usuari. Un formulari pot ser una simple finestra, una finestra MDI, o un diàleg. I en tots els casos, s'ha de tenir present que un formulari és un objecte d'una classe determinada (concretament, cada formulari és una classe derivada de *System.Windows.Forms.Form*), amb la qual cosa presentarà un conjunt de propietats, mètodes i esdeveniments:

Finestra MDI

MDI, de l'anglès *multiple document interface*, són aquelles finestres que poden contenir diverses finestres. En són exemples Adobe Photoshop, o Visual Studio de Microsoft.

- **Propietats**: permeten, per exemple, canviar l'aparença d'un formulari (color, mida, posició, etc.).

- **Mètodes**: exposen el comportament del formulari (mostrar, ocultar, tancar, moure, etc.).

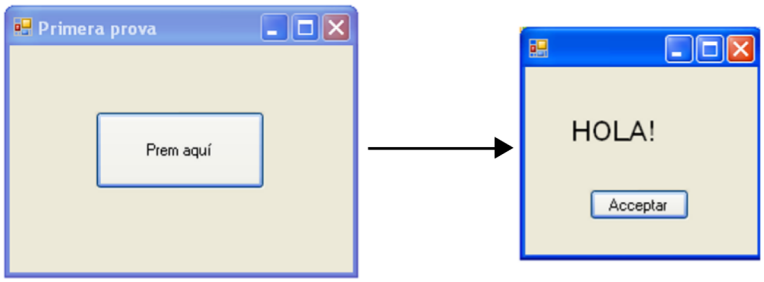
- **Esdeveniments**: permeten interactuar amb el formulari i associar-hi el codi que s'haurà d'executar quan es produeixin aquests esdeveniments (en tancar el formulari, en minimitzar-lo, en moure'l, etc.).

Un formulari conté internament un conjunt de controls (botons, etiquetes, caixes de text, etc.) que ens permetran crear la interfície d'usuari. Així com el formulari té uns esdeveniments associats, cada un dels controls també tindrà els seus esdeveniments propis. Per a tots aquells esdeveniments que

ens interessin, implementarem la funció corresponent, que contindrà el codi que s'haurà d'executar quan es produeixi l'esdeveniment (per exemple, quan l'usuari premi un botó, es mostrarà un missatge per pantalla).

Per entendre el codi generat automàticament per Visual Studio, crearem una primera aplicació de tipus Windows Forms, a la qual afegirem un botó, i en prémer-lo mostrarà un missatge "HOLA!":

Figura 7. Primera aplicació WinForms



Vegem-ne el codi corresponent:

a) Form1.cs: conté el constructor i el codi associat als esdeveniments.

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }

  private void Button1_Click(object sender, EventArgs e)
  {
    MessageBox.Show("HOLA!");
  }
}
```

- En crear el formulari amb Visual Studio, s'ha creat automàticament una classe *Form1*, que deriva de *Form*.
- La classe és *partial*, la qual cosa permet repartir el codi en dos arxius: d'una banda, Form1.cs, on el programador escriurà el codi associat als esdeveniments, i, de l'altra, Form1.Designer.cs, generat automàticament per Visual Studio a partir del disseny que fem del formulari, i que el programador en principi no haurà d'editar mai.
- El constructor fa la crida al mètode *InitializeComponent*, que ha estat generat per Visual Studio, i és on s'inicialitzen tots els controls del formulari.
- El mètode *Button1_Click* serà cridat quan l'usuari premi el botó *Button1*. En aquell moment, apareixerà el missatge "HOLA!" per pantalla.

b) Form1.Designer.cs: conté el codi generat automàticament per Visual Studio (recomanem no modificar-lo manualment).

```
partial class Form1
{
  private IContainer components = null;

  protected override void Dispose(bool disposing)
  {
    if (disposing &amp;amp;&amp;amp; (components != null))
    {
      components.Dispose();
    }
    base.Dispose(disposing);
  }

  private void InitializeComponent()
  {
    this.Button1 = new Button();
    this.SuspendLayout();

    this.Button1.Location = new Point(65, 31);
    this.Button1.Name = "Button1";
    this.Button1.Size = new Size(125, 74);
    this.Button1.TabIndex = 0;
    this.Button1.Text = "Prem aquí!";
    this.Button1.UseVisualStyleBackColor = true;
    this.Button1.Click +=
      new EventHandler(Button1_Click);

    this.AutoScaleDimensions = new SizeF(6F, 13F);
    this.AutoScaleMode = AutoScaleMode.Font;
    this.ClientSize = new Size(263, 153);
    this.Controls.Add(this.Button1);
    this.Name = "Form1";
    this.Text = "Primera prova";
    this.ResumeLayout(false);
  }

  private Button Button1;
}
```

- El mètode *Dispose* ha estat creat per Visual Studio, i conté el codi per a alliberar els recursos del formulari.

- El mètode *InitializeComponent* també ha estat creat per Visual Studio. En el codi, creem una instància del botó, n'inicialitzem les propietats i assig-

nem el mètode *Button1_Click* a l'esdeveniment *Click* del botó. Tot seguit, inicialitzem les propietats del formulari, com per exemple la mida o títol de la finestra.

- El formulari conté una variable per cada un dels seus controls. En aquest cas, com que només hi ha un botó, conté una única variable cridada *Button1*, que és de la classe *Button*.

5.1. Implementació d'esdeveniments

La implementació dels esdeveniments de C# està basada en l'ús de delegats, que permeten passar funcions com a paràmetres a altres funcions. Concretament, un delegat permet al programador encapsular una referència a un mètode dins d'un objecte delegat. Posteriorment, l'objecte delegat podrà ser passat com a paràmetre a un codi que s'encarregarà de fer la crida al mètode referenciat, sense necessitat d'haver de saber quin és el mètode en qüestió en temps de compilació.

Equivalència

A C/C++ l'equivalent als delegats serien els punters a funcions.

5.1.1. Delegats

Un delegat permet emmagatzemar una referència a qualsevol mètode que compleixi un contracte, que ve definit pel tipus i nombre de paràmetres, i pel tipus del valor de retorn de la funció. A tall d'exemple, el delegat següent permetrà referenciar qualsevol funció que rebi dos enters com a paràmetres, i retorni un altre enter:

```
public delegate int ElMeuDelegat (int valor1, int valor2);
```

A l'exemple següent, es defineix i utilitza un delegat. En executar l'aplicació, es mostraria per la pantalla el valor "8" corresponent al valor de 2^3:

```
class Program
{
  public delegate int ElMeuDelegat(int valor1, int valor2);

  public static int suma(int valor1, int valor2)
  {
    return valor1 + valor2;
  }

  public static int potencia(int base, int exponent)
  {
    return (int)Math.Pow(base, exponent);
  }

  public static void calcul(ElMeuDelegat f, int a, int b)
```

```
  {
    Console.WriteLine(f(a, b));
  }

  static void Main(string[] args)
  {
    ElMeuDelegat delegat = new ElMeuDelegat(potencia);
    calcul(delegat, 2, 3);
  }
}
```

5.1.2. Funcions gestores d'esdeveniments

Les funcions associades als esdeveniments es gestionen mitjançant delegats amb el contracte següent:

```
delegate void NombreEventHandler (object Sender, EventArgs e);
```

- La funció no retorna res (*void*).
- El primer paràmetre és l'objecte que ha generat l'esdeveniment.
- El segon paràmetre emmagatzema les dades que poden ser utilitzades en la funció gestora de l'esdeveniment. Pot tractar-se d'un objecte de la classe *EventArgs* o d'una classe derivada que permeti emmagatzemar informació addicional en funció del tipus d'esdeveniment.

En l'exemple següent, mostrem diverses funcions gestores d'esdeveniments:

```
private void Form2_Load(object sender, EventArgs e)
{
}

private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
  MessageBox.Show(e.KeyChar.ToString());
}

private void textBox1_MouseClick(object sender,MouseEventArgs e)
{
  MessageBox.Show(e.X + " " + e.Y);
}
```

- La primera correspon a l'esdeveniment *Load* del formulari, que s'executa una única vegada, just abans que el formulari es mostri per pantalla per primera vegada. Aquest esdeveniment és molt utilitzat, ja que és el que afegeix Visual Studio automàticament per defecte en fer doble clic en un formulari qualsevol.

- La segona funció correspon a prémer una tecla en un control *TextBox*, en què es mostra per pantalla la tecla que s'ha pitjat.

- La tercera funció correspon a la pulsació d'un botó del ratolí sobre un control *TextBox*, que mostra per pantalla la posició (X,Y) on s'ha esdevingut la pulsació.

5.2. Controls

En un formulari podem arrossegar múltiples elements de la paleta de controls; val a dir que, si som precisos, la paleta inclou tant contenidors com components i controls. Vegem, tot seguit, les diferències entre els uns i els altres:

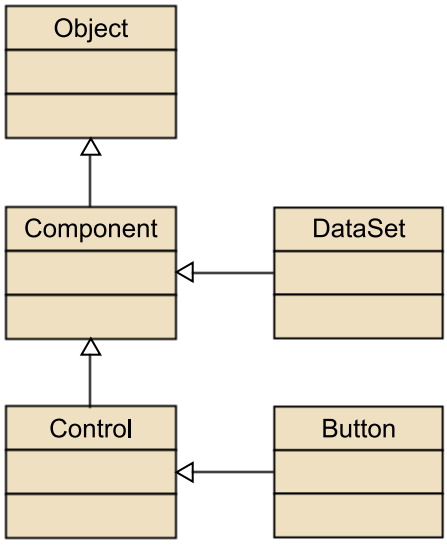
- Un contenidor és un objecte que pot contenir múltiples components. Per exemple, a la paleta de controls disposem dels contenidors *Panel*, *GroupBox* o *TabControl*.

- Un component és una subclasse de *Component* que ofereix una interfície clarament definida i està dissenyada per a ser reutilitzada per terceres aplicacions, però que no té per què mostrar cap interfície gràfica. Un exemple seria el component *DataSet*, que permet emmagatzemar dades, però no mostra res per pantalla (per a això, s'han d'utilitzar altres controls).

- Un control és una classe derivada de *Control*, i és un component que a més té interfície gràfica. Un exemple seria el control *Button*, que mostra un botó.

A la figura següent, mostrem de manera molt simplificada la jerarquia de classes corresponent al component *DataSet* i al control *Button*:

Figura 8. Jerarquia simplificada de components i controls



.NET Framework proporciona multitud de components i controls que ens permeten desenvolupar interfícies gràfiques de manera ràpida i senzilla. A més, en cas que no trobi el control necessari, podem implementar els nostres propis controls personalitzats.

Controls addicionals

A Internet podem trobar multitud de controls addicionals, i n'hi ha tant de gratuïts com de comercials.

A continuació, es repassen breument els elements principals de la paleta de controls de Visual Studio: contenidors, controls, components i diàlegs.

5.2.1. Contenidors

A la paleta de controls de Visual Studio podem trobar els contenidors següents:

- ***Panel (plafó)***: permet agrupar un conjunt de controls, i tractar-los de manera simultània. A tall d'exemple, ens serviria per a poder mostrar o ocultar tot un conjunt de controls per pantalla.

- ***SplitContainer***: són dos plafons (*panel*) separats per una barra movible (anomenada *splitter*) que l'usuari pot fer servir per a redimensionar la mida dedicada a cada un dels plafons.

- ***GroupBox***: és molt similar al plafó, i la diferència principal és l'aparença que mostra per pantalla.

- ***TabControl***: permet allotjar controls en diferents grups, i l'usuari pot accedir als uns o als altres mitjançant diferents pestanyes.

5.2.2. Controls

Els controls habituals de la paleta de Visual Studio són:

- ***Label***: és una etiqueta on podem mostrar text per pantalla.

- ***TextBox***: és una caixa de text en què l'usuari pot escriure.

- ***PictureBox***: permet mostrar una imatge.

- ***Button***: és un botó que pot prémer l'usuari.

- ***ListBox***: llista d'elements dels quals l'usuari en pot seleccionar un o més.

- ***ComboBox***: similar al *ListBox*, amb la diferència que és una llista desplegable i l'usuari només pot seleccionar un element.

- ***CheckBox***: element que l'usuari pot marcar com a seleccionat, o no. En cas d'haver-hi diversos *CheckBox* un sota l'altre, funcionen de manera independent, de manera que l'usuari en pot seleccionar tants com desitgi.

- ***RadioButton***: element similar al *CheckBox*, però que està dissenyat perquè l'usuari en seleccioni únicament un d'entre un conjunt d'opcions. A tall d'exemple, es pot utilitzar per preguntar a l'usuari si està solter o casat, de manera que haurà de seleccionar una opció o l'altra, però no totes dues alhora.

- ***TreeView***: permet mostrar un conjunt d'elements jeràrquics dins d'una estructura de tipus arbre.

- ***DataGridView***: control molt potent, que permet mostrar i gestionar dades en forma de taula. És molt útil per a fer el manteniment d'una taula de la base de dades.

- ***MenuStrip***: permet crear un menú de manera molt senzilla. Aquest menú pot contenir al seu torn altres submenús.

- ***ContextMenuStrip***: permet crear un menú contextual i associar-lo a un conjunt d'elements. D'aquesta manera, quan l'usuari premi el botó dret sobre un element determinat, es mostrarà el menú contextual corresponent.

5.2.3. Components

Així mateix, entre els elements de la paleta de controls de Visual Studio, trobem els components següents:

- ***FileSystemWatcher***: permet vigilar el sistema de fitxers i reaccionar davant de modificacions que hi tinguin lloc. A tall d'exemple, podem configurar-lo perquè ens avisi quan s'elimini o modifiqui un arxiu en un directori determinat.

- ***MessageQueue***: ofereix accés a un servidor de missatgeria de cues, com per exemple MSMQ (*Microsoft Message Queuing*).

- ***SerialPort***: ens permet establir comunicacions per un port sèrie.

- ***Timer***: llança un esdeveniment cada cert període de temps.

5.2.4. Diàlegs

També disposem d'un conjunt de diàlegs prefabricats, que són molt útils per a tasques quotidianes com ara escollir la ruta d'un arxiu, o seleccionar un color d'una paleta de controls. Concretament, disposem dels controls següents:

- ***ColorDialog***: permet seleccionar un color d'una paleta de colors.
- ***FolderBrowserDialog***: permet seleccionar un directori.
- ***FontDialog***: permet seleccionar un tipus de font.
- ***OpenFileDialog***: permet obrir un arxiu.
- ***SaveFileDialog***: permet enregistrar en un arxiu.
- ***PrintDialog***: permet imprimir per una impressora.
- ***PrintPreviewDialog***: mostra una vista preliminar d'impressió.
- ***PageSetupDialog***: permet establir les mides de la pàgina, els marges, etc., abans d'imprimir.

En cas que necessitem un diàleg que no es correspongui amb cap dels anteriors, crearem els nostres propis diàlegs a mida. Per a això, n'hi ha prou d'afegir nous formularis al projecte, i quan l'aplicació estigui en execució, anirem mostrant un formulari o un altre segons ens convingui.

Finalment, podem mostrar els diàlegs de dues maneres:

- **Modal**: exigeix que l'usuari respongui al diàleg per a poder continuar amb l'execució.

- **No modal**: no exigeix que l'usuari respongui al diàleg, de manera que es comporta com una finestra independent. L'usuari pot canviar entre una finestra o l'altra sense cap problema.

Per a mostrar un diàleg de manera modal, fem servir el mètode *ShowDialog*, mentre que per a mostrar-lo de manera no modal, s'ha d'emprar el mètode *Show*.

6. ASP.NET

En aquest apartat, introduïm la tecnologia ASP.NET, que permet crear aplicacions web. Després de la visió general sobre ASP.NET i un repàs d'alguns dels controls més utilitzats, oferim una introducció a AJAX, que permet millorar l'experiència d'usuari en les aplicacions web.

6.1. Una introducció a ASP.NET

ASP.NET és un *framework* per a la creació d'aplicacions web, en el qual es pot programar en qualsevol dels llenguatges de .NET. Va aparèixer l'any 2002 i és la tecnologia successora d'Active Server Pages (ASP), que existeix des de 1996.

ASP

Les pàgines ASP contenen *scripts* programats habitualment en VBScript.

ASP.NET ofereix múltiples avantatges en comparació amb l'antiga ASP:

- ASP.NET s'integra totalment amb .NET i les seves pàgines es poden programar en qualsevol dels llenguatges de .NET, fent ús de la programació orientada a esdeveniments.
- ASP.NET ofereix un conjunt de controls molt més ric.
- ASP era interpretat, mentre que ASP.NET és compilat. Això ofereix múltiples avantatges, com un rendiment molt millor, i una depuració molt més potent.
- La configuració i desplegament d'aplicacions ASP.NET és molt més senzilla, ja que la configuració té lloc en únic arxiu text, i per a fer-ne el desplegament n'hi ha prou de copiar els arxius en el directori corresponent.

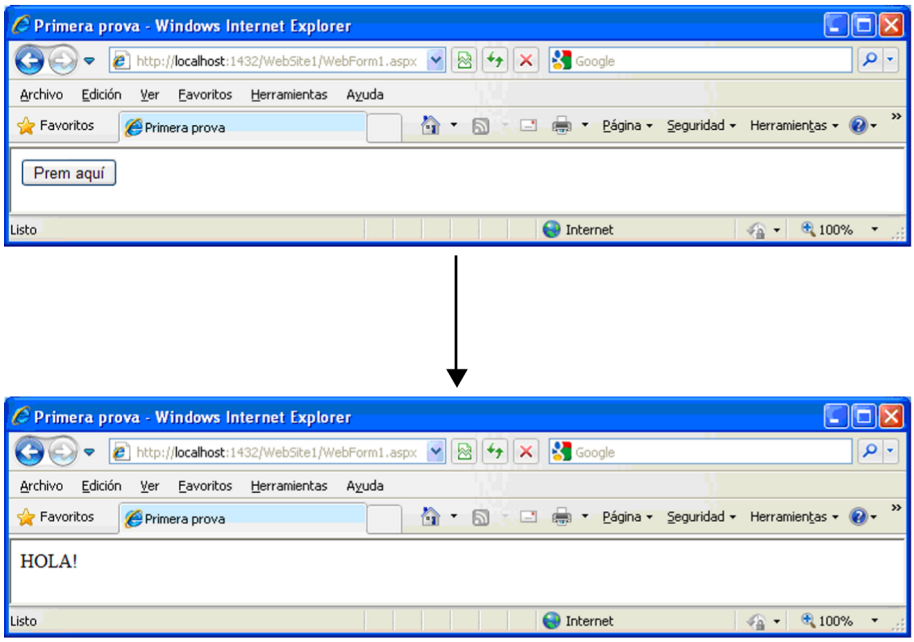
Les pàgines ASP.NET es denominen *web forms* (formularis web), i són arxius amb extensió .aspx. Aquests arxius estan formats bàsicament per marques XHTML estàtic, i també per marques ASPX que li donen el comportament dinàmic. Un formulari web és una classe derivada de *System.Web.UI.Page*, amb un conjunt de propietats, mètodes i esdeveniments:

- **Propietats**: permeten, per exemple, canviar l'aparença d'un formulari (el títol, color de fons, estils CSS, etc.).
- **Mètodes**: exposen el comportament del formulari.
- **Esdeveniments**: permeten interactuar amb el formulari i associar-hi el codi que s'haurà executar quan es produeixin aquests esdeveniments.

El disseny i la programació de formularis web són molt similars a WinForms, de manera que un formulari conté un conjunt de controls que formen la interfície d'usuari, i aquests responen a una sèrie d'esdeveniments a què s'associa el codi corresponent.

La pàgina ASP.NET següent conté un botó, que en prémer-lo mostra el missatge "HOLA!":

Figura 9. Primera aplicació ASP.NET



El codi corresponent és el següent:

a) WebForm1.aspx: conté el codi XHTML de la pàgina, barrejat amb codi ASP.NET.

```
<%@ Page Language = "C#" AutoEventWireup="true"
                    CodeFile="WebForm1.aspx.cs" Inherits="WebForm1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Primera prova</title>
</head>
<body>
<form id="form1" runat="server">
<asp:Button ID="Button1" runat="server" Text="Prem aquí."
      onclick="Button1_Click" />
<asp:Label ID="Label1" runat="server"></asp:Label>
</form>
```

```
</body>
</html>
```

- L'element *Page* indica que la pàgina utilitza el llenguatge C#, que deriva de la classe *WebForm1*, i que el codi C# està a l'arxiu WebForm1.aspx.cs.

- L'element *DOCTYPE* indica el DTD de XHTML utilitzat a la pàgina, que és concretament XHTML Transitional.

- L'element *head* fa referència a la capçalera de la pàgina XHTML, on s'indica, per exemple, el títol de la pàgina.

- Al cos de la pàgina, hi ha un formulari que conté un botó i una etiqueta. En tots dos controls podem veure *runat* = "*server*", que indica que es processen en el servidor.

- El botó té associada la funció *Button1_Click* a l'esdeveniment *onclick*, que s'executarà quan l'usuari premi el botó.

b) WebForm1.aspx.cs: conté el codi dels esdeveniments de la pàgina.

```
public partial class WebForm1 : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }

  protected void Button1_Click(object sender, EventArgs e)
  {
    Button1.Visible = false;
    Label1.Text = "HOLA!";
  }
}
```

- La classe del formulari deriva de *System.Web.UI.Page*.

- El mètode *Page_Load* serà cridat cada vegada que es carrega la pàgina.

- El mètode *Button1_Click* serà cridat quan l'usuari premi el botó *Button1*. En aquell moment, ocultarem el botó i mostrarem "HOLA!" per pantalla.

c) I finalment, el codi HTML que ha rebut el navegador:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
  <title>Primera prova</title>
</head>
<body>
<form id="form1" method="post" action="WebForm1.aspx">
<input type="hidden" id="__VIEWSTATE" value="..." />
  <span id="Label1">HOLA!</span>
</form>
</body>
</html>
```

- El formulari *form1* és de tipus *post* per defecte, i l'*action* apunta a la pàgina *WebForm1.aspx*, que serà on s'enviarà la sol·licitud quan l'usuari premi el botó.

- Tota pàgina conté, per defecte, un element *VIEWSTATE* de tipus *hidden*. Aquest element permet emmagatzemar les dades entrades per l'usuari en un formulari, i així l'estat de la pàgina pot persistir entre diverses interaccions de l'usuari.

Esdeveniment *Load* del formulari

A l'exemple següent, es mostra un ús típic de l'esdeveniment *Load*:

```
private void Form2_Load(object sender, EventArgs e)
{
  if (!Page.IsPostBack)
     Label1.Text = "Benvingut";
}
```

- L'esdeveniment *Load* s'executa cada vegada que es carrega la pàgina. Per tant, com l'*action* del formulari fa referència a la pàgina mateixa, això fa que, quan l'usuari prem el botó, el primer esdeveniment que s'executa és *Page_Load*, i tot seguit el *Button1_Click*.

- La propietat *Page.IsPostBack* és molt útil, i permet diferenciar entre la primera vegada que s'executa l'esdeveniment *Page_Load*, i la resta d'ocasions. En l'exemple anterior, la primera vegada que es carrega la pàgina es mostra el text "Benvingut".

6.1.1. Controls

ASP.NET ofereix un ampli repertori de controls que podem utilitzar als formularis web. A continuació, en repassem alguns dels principals.

Controls estàndard

Els controls estàndard són els següents:

- ***Button***: és un botó que, en prémer-lo, envia la petició corresponent al servidor.
- ***CheckBox***: permet marcar o desmarcar una certa opció.
- ***DropDownList***: molt semblant al control ComboBox de WinForms, on l'usuari selecciona un valor d'un desplegable.

- ***Image***: permet mostrar una imatge.
- ***Label***: permet mostrar un text.
- ***ListBox***: permet seleccionar un o més valors d'una llista.
- ***RadioButton***: element similar al *CheckBox*, però amb la diferència que si n'hi ha més d'un, l'usuari només en pot seleccionar un.
- ***TextBox***: permet que l'usuari escrigui un text.

Controls de dades

Els controls de dades són un conjunt de controls encarregats de la connexió amb l'origen de dades, i també de la consulta i manipulació de les dades esmentades:

- ***SqlDataSource***: permet connectar amb qualsevol base de dades relacional a què es pugui accedir mitjançant un proveïdor ADO.NET, ja sigui ODBC, OLE DB, SQL-Server o Oracle. El seu objectiu és reemplaçar el codi ADO.NET necessari per a establir una connexió i definir la sentència SQL a executar.

- ***XmlDataSource***: permet enllaçar amb un document XML, i fer consultes mitjançant *XPath*.

- ***GridView***: és el successor del control *DataGrid* d'ASP.NET, i permet editar dades en forma tabular. De manera senzilla, podem modificar o eliminar dades, seleccionar una fila determinada, paginar, etc.

- ***DetailsView***: control similar al *GridView*, però que només permet treballar amb un únic registre alhora.

Controls de validació

Els controls de validació són un conjunt de controls que faciliten la validació de les dades entrades per l'usuari en un formulari. Per a més seguretat, aquests controls fan una primera validació en el navegador per mitjà de Javascript, i posteriorment una segona validació en el servidor. Són els següents:

- ***RequiredFieldValidator***: assegura que l'usuari no pugui deixar en blanc un camp obligatori.
- ***RangeValidator***: comprova que el valor entrat per l'usuari estigui dins d'un rang de valors determinat.
- ***RegularExpressionValidator***: comprova que el valor entrat per l'usuari compleixi amb una expressió regular determinada.

Controls de connexió

Els controls de connexió són un conjunt de controls encarregats de les funcionalitats d'autenticació d'usuaris, i de l'entrada de l'usuari i contrasenya, creació d'usuaris, canvi de contrasenya, restauració de la contrasenya en cas de pèrdua, etc.

Controls de navegació

Els controls de navegació són els controls que faciliten la navegació de l'usuari per l'aplicació:

- ***SiteMapPath***: mostra un conjunt d'enllaços que representen la pàgina de l'usuari i el camí jeràrquic de retorn a la pàgina arrel del web.

- ***Menu***: permet afegir un menú a l'aplicació.

- ***TreeView***: permet representar un conjunt d'elements organitzats jeràrquicament en forma d'arbre.

Controls de *webparts*

Nou conjunt de controls disponibles a partir d'ASP.NET 2.0, que permet a l'usuari personalitzar el contingut i aspecte de les pàgines web. Alguns d'aquests controls són els següents:

- ***WebPartManager***: gestiona tots els *webparts* d'una pàgina.
- ***CatalogZone***: permet crear un catàleg de controls *webpart* que l'usuari pot seleccionar i afegir a la pàgina.
- ***EditorZone***: permet a l'usuari editar i personalitzar els controls *webpart* d'una pàgina.
- ***WebPartZone***: conté tots els controls *webpart* que s'hagin inclòs a la pàgina.

Controls HTML

Els controls HTML són controls que permeten treballar amb elements simples HTML com ara taules, enllaços, imatges, botons, etc. La diferència amb els controls estàndard d'ASP.NET és que els HTML generen el mateix codi HTML, independentment del navegador del client, mentre que els ASP.NET generen un codi HTML que depèn del navegador del client.

Controls d'usuari

Finalment, també podem crear els nostres propis controls d'usuari de manera molt senzilla. Per a això, cal crear un projecte web de tipus ASP.NET, i afegir-hi un nou element de tipus *Web User Control*. Això afegeix al projecte un arxiu

.ascx, que serà on crearem el nou control d'usuari. Un cop fet això, en compilar el projecte obtindrem un arxiu amb extensió .dll corresponent al nou control d'usuari, que podrem afegir a la paleta de controls de Visual Studio.

6.2. AJAX

Històricament, les aplicacions d'escriptori sempre han ofert una interacció amb l'usuari molt més rica que les aplicacions web. En contrapartida, les aplicacions web no s'han d'instal·lar, i ens asseguren que els usuaris sempre tenen l'última versió de l'aplicació, ja que tan sols cal actualitzar-la una vegada en el servidor.

En canvi, amb l'aparició d'AJAX el tema ha canviat, ja que és possible crear aplicacions web que ofereixin una rica interacció amb l'usuari. En realitat, les tecnologies que han habilitat l'aparició d'AJAX existeixen des de fa més d'una dècada, ja que es tracta bàsicament de Javascript i XML. Tanmateix, no és fins fa poc que se'ls ha tret tot el suc amb aplicacions com Google Gmail, Microsoft Outlook Web Access, Flickr, etc.

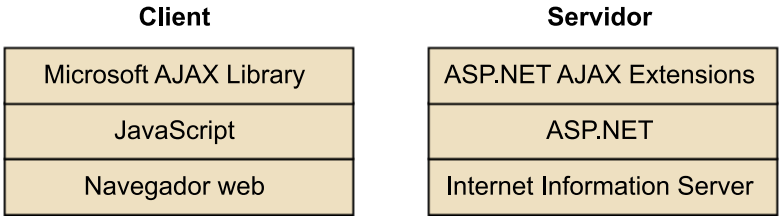
6.2.1. Una introducció a AJAX

En desenvolupar una aplicació AJAX, en lloc de dissenyar-la com una sèrie de pàgines enllaçades (en què en acabar de processar una pàgina, simplement es mostra la següent), es fa servir Javascript per a mantenir una comunicació asíncrona amb el servidor web, i així poder actualitzar parts de les pàgines de manera dinàmica.

Per exemple, si l'usuari selecciona un país en una llista desplegable de països, s'estableix, de manera transparent per a l'usuari, una comunicació amb el servidor en què se li envia el país i aquest retorna una llista de ciutats del país esmentat. Posteriorment, podem sol·licitar a l'usuari que seleccioni una ciutat d'aquest país en concret (això haurà tingut lloc sense haver recarregat la pàgina completa, la qual cosa hauria estat el procediment habitual).

ASP.NET AJAX és el nom de la solució AJAX de Microsoft, i fa referència a un conjunt de tecnologies de client i de servidor que faciliten l'ús de tecnologia AJAX en .NET. La figura següent mostra, de manera esquemàtica, els components de les aplicacions AJAX en .NET:

Figura 10. Components d'AJAX en el client i en el servidor



D'una banda, en el client disposem de Microsoft AJAX Library, una llibreria Javascript que funciona en múltiples navegadors, i que facilita el desenvolupament amb Javascript. L'utilitzarem per a interactuar amb el DOM, actualitzar porcions de la pàgina dinàmicament, realitzar comunicacions asíncrones amb el servidor, etc. De fet, el gran avantatge de Microsoft AJAX Library és que ofereix una capa orientada a objectes d'alt nivell, que evita tota la farragosa programació JavaScript a baix nivell.

D'altra banda, en el servidor disposem d'ASP.NET AJAX Extensions, que està construït per damunt dels controls i classes d'ASP.NET, i ens permet fer ús de Microsoft AJAX Library.

6.2.2. Actualitzacions parcials de pàgines

El control *UpdatePanel* és un control contenidor que permet marcar les porcions d'una pàgina que es vulguin actualitzar asíncronament. Quan algun dels controls de dins de l'*UpdatePanel* genera una tramesa al servidor (per exemple la pulsació d'un botó), podem iniciar una comunicació asíncrona amb el servidor i actualitzar únicament aquesta porció de pàgina. El fet que sigui asíncron vol dir que l'usuari podrà continuar utilitzant la pàgina i interactuar amb la resta de controls mentre esperem rebre la resposta del servidor. Quan en tinguem la resposta, actualitzarem la porció de pàgina corresponent d'una manera molt suau a la vista, de manera que l'usuari no notarà cap parpelleig ni recàrrega de la pàgina.

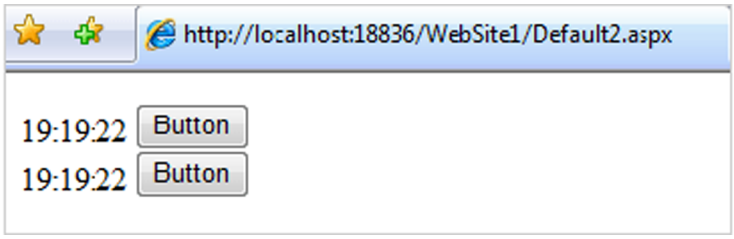
Tot seguit, s'indiquen els passos a seguir per a realitzar un exercici molt simple amb AJAX, i començar a entendre'n el funcionament més bàsic:

1) Crear un projecte web de tipus ASP.NET.
2) Afegir al projecte una pàgina de tipus *Web Form*.
3) Afegir a la pàgina un control de tipus *ScriptManager*, que cal tenir en tota pàgina amb AJAX, ja que és l'encarregat de gestionar el refrescament parcial de la pàgina.
4) Afegir a sota un control *Label*, i al costat un control *Button*.
5) Afegir a sota un control *UpdatePanel*, i dins d'aquest control, afegir un nou control *Label*, i al costat, un altre control *Button*.
6) Fer doble clic sobre el formulari, i en l'esdeveniment *Page_Load* afegir el codi següent:

```
protected void Page_Load(object sender, EventArgs e)
{
  Label1.Text = DateTime.Now.ToLongTimeString();
  Label2.Text = DateTime.Now.ToLongTimeString();
}
```

En executar la pàgina, hauríem d'obtenir una cosa semblant a això:

Figura 11. Exemple senzill amb un *UpdatePanel*



A la pàgina, tenim dues etiquetes i dos botons; l'etiqueta i el botó de la part inferior estan ubicats dins del control *UpdatePanel*. Per tant, en prémer el botó superior observarem que té lloc una petició normal al servidor, mitjançant la qual es refresca la pàgina sencera, que aleshores mostra les dues hores actualitzades.

En canvi, en prémer el botó inferior, s'inicia una comunicació asíncrona amb el servidor, que retorna el valor de l'hora inferior, i únicament aquesta és actualitzada pel navegador. Haurem pogut observar com, en aquest últim cas, l'actualització ha estat molt suau, i no s'ha produït cap recàrrega de la pàgina.

Reflexió

Les possibilitats de Microsoft AJAX Library són extenses i queden fora de l'abast d'aquest llibre. Per a més informació, consulteu la bibliografia recomanada.

6.2.3. AJAX Control Toolkit

AJAX Control Toolkit és un projecte de codi obert integrat per empleats de Microsoft i també per altres empreses, l'objectiu de les quals és crear un conjunt ric de controls que facin servir AJAX.

Sobre AJAX Control Toolkit

Podeu trobar tota la informació sobre el projecte AJAX Control Toolkit (accessible en línia) en la seva pàgina web.

En aquesta pàgina podeu consultar una llista de tots els controls del Toolkit, juntament amb una demostració de com funciona i informació explicativa.

Toolkit incorpora tant nous controls com ampliacions que s'associen a controls ja existents, i afegeixen diverses característiques AJAX.

Vegem alguns dels controls de Toolkit:

- ***Animation***: permet fer múltiples animacions en funció de les accions de l'usuari. És possible moure un element, canviar-lo de mida, ocultar-lo, etc.

- ***Accordion***: permet tenir un conjunt de plafons, i mostrar-ne un o un altre amb un efecte similar al que fa servir Microsoft Outlook.

- ***AlwaysVisibleControl***: permet que un control estigui sempre visible en un posició fixa de pantalla, independentment que l'usuari faci *scroll* cap amunt o cap avall.

- ***Calendar***: apareix una finestra emergent amb un calendari, i permet a l'usuari seleccionar una data.

- ***CollapsiblePanel***: control similar a l'*Accordion*, però que únicament treballa amb un plafó, que es mostra o oculta clicant sobre un botó. Resulta molt útil per a ocultar una porció de pàgina que a vegades l'usuari no voldrà consultar.

- ***ConfirmButton***: mostra un diàleg emergent que ens sol·licita confirmar una acció.

- ***ResizableControl***: permet que un element pugui ser canviat de mida per l'usuari.

- ***Tabs***: permet mostrar el contingut de la pàgina classificat en diferents pestanyes.

7. WPF

En aquest apartat s'introdueix la nova tecnologia WPF.

WPF (Windows Presentation Foundation) està basat en XML, i permet crear potents interfícies d'usuari, tant per a aplicacions d'escriptori com per a entorns web (WPF Browser o Silverlight).

WPF permet dissenyar interfícies gràfiques d'usuari més espectaculars que WinForms, però no està pensat per a aplicacions que requereixin un ús intensiu de gràfics o 3D, com, per exemple, jocs. En aquest camp, les tecnologies més adequades són encara DirectX o OpenGL.

WPF va aparèixer l'any 2006, com una part de .NET Framework 3.0 i Windows Vista. WPF és l'evolució de Windows Forms.

Tot i que WPF està pensat per a crear aplicacions d'escriptori, hi ha variants que l'utilitzen en entorns web:

- **XBAP** o **WPF Browser**: són aplicacions WPF que s'executen dins d'un navegador web (actualment, Internet Explorer o Firefox) en entorn Windows. Presenten totes les característiques de WPF, però cal tenir en compte que, en ser dins d'un navegador, les aplicacions estan en un entorn de seguretat restringit.

- **Silverlight**: és un subconjunt de WPF per a la creació d'aplicacions web. A diferència de XBAP, Silverlight està dissenyat per a ser multiplataforma. Una altra diferència és que Silverlight és molt lleuger, amb la qual cosa el connector necessari és molt més ràpid d'instal·lar.

Silverlight per a Linux

Ja està disponible Moonlight, que és una implementació de codi obert de Silverlight per a sistemes Linux.

7.1. Característiques

Algunes de les característiques més interessants de WPF són les següents:

- WPF integra múltiples tecnologies, com per exemple processament gràfic 2D, 3D, vídeo, àudio, etc.

- WPF utilitza internament DirectX, i per això es beneficia de totes les característiques relatives als gràfics vectorials, les característiques avançades de visualització, tècniques 3D, acceleració per maquinari, etc.

- Les interfícies gràfiques de WPF es poden definir de manera declarativa amb el llenguatge XAML (eXtensible Application Markup Language), la qual cosa permet separar la interfície gràfica de la lògica de l'aplicació.

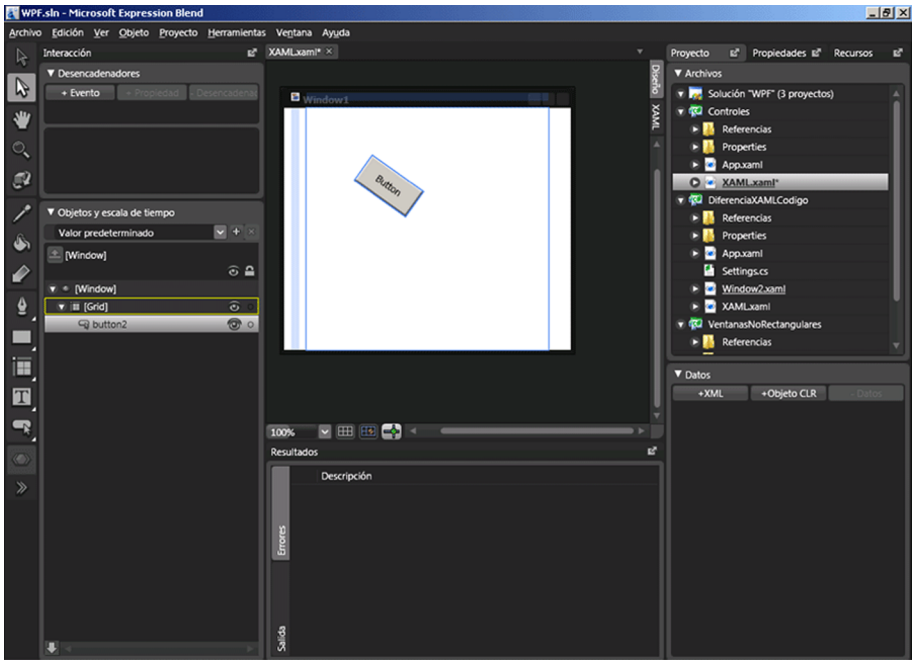
- WPF és interoperable amb WinForms i amb Win32.

- WPF permet una fàcil implantació, ja que, en ser .NET, n'hi ha prou de copiar els assemblatges.

7.1.1. Eines de desenvolupament

Tot i que l'entorn de programació WPF és Visual Studio, Microsoft ha introduït l'entorn de disseny Expression Blend pensat per als dissenyadors gràfics; es pot utilitzar sense tenir coneixements de programació.

Figura 12. Expression Blend



De fet, Expression Blend forma part del paquet d'eines de disseny anomenat Microsoft Expression Studio, que inclou:

- **Expression Web**: eina per a la creació d'interfícies web amb HTML. És l'aplicació successora de FrontPage, projecte que ja ha estat abandonat per part de Microsoft.

- **Expression Blend**: eina per a la creació d'interfícies gràfiques d'usuari en XAML per a aplicacions WPF o Silverlight.

- **Expression Design**: eina per a la creació de gràfics vectorials. Ofereix una bona integració amb Expression Blend gràcies a la importació/exportació de codi XAML.

- **Expression Media**: és un organitzador de contingut multimèdia (fotos, vídeos o àudio). Permet categoritzar i afegir metadades als diferents arxius.

- **Expression Encoder**: aplicació per a la recodificació de vídeos. Permet elegir un nivell de qualitat predefinit, un còdec i format de sortida, i previsualitzar-ne el resultat.

7.1.2. XAML

XAML és un llenguatge XML que permet descriure una interfície gràfica WPF. Però no totes les aplicacions WPF necessàriament han d'utilitzar XAML, ja que és possible definir interfícies gràfiques de manera procedimental (amb codi font en algun llenguatge de .NET). Vegem-ne un exemple amb un botó:

```
Button b = new Button();
b.Name = "button1";
b.Content = "Clica aquí";
b.Height = 23;
```

El codi anterior és equivalent al fragment de XAML següent:

```
<Button Height="23" Name="button1">
  Clica aquí
</Button>
```

Un dels motius per utilitzar XAML és que permet separar la interfície gràfica de la lògica d'aplicació. Aquesta separació permet modificar la visualització de la interfície sense afectar la lògica de l'aplicació i viceversa.

De tota manera, tot i que XAML és un llenguatge declaratiu, a l'hora de compilar una aplicació WPF, el codi XAML es converteix en una representació binària anomenada BAML (*Binary Application Markup Language*), més eficient i compacta, que s'inclou com a recurs dins de l'assemblatge final.

Hi ha diverses variants de XAML; algunes de les més conegudes són:

- XAML per a interfícies gràfiques WPF.

- XAML per a definir el format de documents electrònics XPS.
- XAML per a Silverlight.
- XAML per a WWF, utilitzat per a definir els fluxos de treball.

Format XPS

De l'anglès *XML Paper Specification*. És un format de documents desenvolupat per Microsoft, i similar a PDF. Windows Vista disposa d'una impressora XPS, i també d'un visor de XPS integrat a Internet Explorer.

7.1.3. Window

L'element principal d'una aplicació WPF és la finestra, representada per la classe *Window*. En crear un projecte WPF en Visual Studio, es crea automàticament un fitxer *xaml* amb un element Window:

```
<Window x:Class="Layout.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="300" Width="300">
<!-- Contingut de la finestra -->
</Window>
```

En WPF, els controls no estan subjectes a una resolució de pantalla concreta, de manera que podem escalar una interfície sense perdre qualitat de visualització, i mantenint la distribució dels controls. Per a organitzar els controls, s'utilitzen uns contenidors anomenats plafons, que hi distribueixen els controls segons un patró determinat.

7.1.4. Controls contenidors

A continuació, veurem alguns dels principals controls contenidors de WPF, com són el *Grid*, *Canvas* i *StackPanel*.

Grid

El *Grid* és un contenidor molt flexible pel que fa a l'organització dels controls, ja que permet definir diverses files i columnes en forma de taula, on posteriorment s'ubiquen els diferents controls.

Al principi, hi ha una secció *RowDefinitions* i *ColumnDefinitions*, on es defineixen el nombre de files i columnes, i també les mides o proporcions que el caracteritzen. Tot seguit, ubiquem cada control a la fila o columna que correspongui, amb les propietats *Grid.Row* i *Grid.Column*. A més, amb les propietats *RowSpan* i *ColumnSpan* podem indicar si un control ocupa més d'una fila o columna:

Proporcions

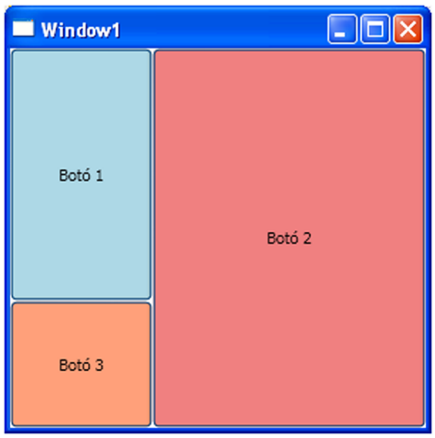
Les proporcions s'indiquen amb el caràcter asterisc (*). En l'exemple, la primera fila té una alçada de 2 *, i això vol dir que serà el doble d'alta que la resta.

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="2*"/>
```

```
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="100"/>
<ColumnDefinition />
</Grid.ColumnDefinitions>
<Button Name="Button1" Content="Boto1" Grid.Row="0"
      Grid.Column="0" Background="LightBlue"/>
<Button Name="Button2" Content="Boto2" Grid.Row="0"
      Grid.Column="1" Background="LightCoral"
      Grid.RowSpan="2"/>
<Button Name="Button3" Content="Boto3" Grid.Row="1"
      Grid.Column="0" Background="LightSalmon"/>
</Grid>
```

En executar l'aplicació, podem comprovar que el botó 2 ocupa les dues files a causa del *RowSpan*, i el botó 1 és el doble d'alt que el botó 3:

Figura 13. Exemple de plafó Grid



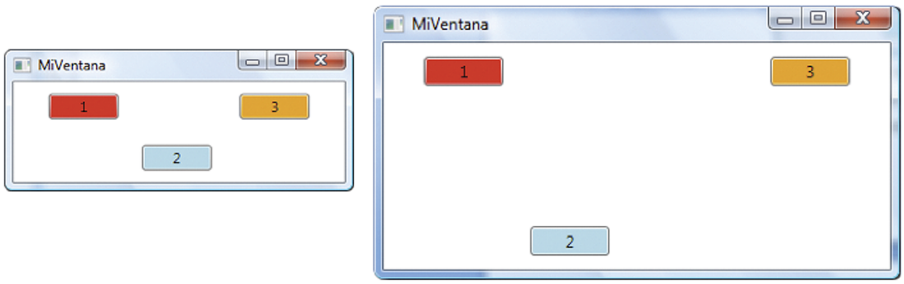
Canvas

En un plafó de tipus *Canvas* els controls se situen en relació amb la distància a la que es troben d'alguna de les cantonades del plafó. Vegem-ne un exemple:

```
<Canvas Name="canvas1">
<Button Canvas.Left="30" Canvas.Top="10"
      Background="Red" Width="60">1</Button>
<Button Canvas.Left="110" Canvas.Bottom="10"
      Background="LightBlue" Width="60">2</Button>
<Button Canvas.Right="30" Canvas.Top="10"
      Background="Orange" Width="60">3</Button>
</Canvas>
```

En executar l'aplicació, podem comprovar com es mantenen les distàncies a les vores independentment de la mida de la finestra:

Figura 14. Exemple de plafó Canvas



StackPanel

El plafó *StackPanel* distribueix seqüencialment els controls ubicats dins seu en forma de pila. L'exemple següent fa servir un *StackPanel* per a distribuir diferents botons:

```
<StackPanel Name="stack1" Orientation="Vertical">
  <Button Background="Red">      1</Button>
  <Button Background="Orange">   2</Button>
  <Button Background="Yellow">   3</Button>
  <Button Background="Blue">     4</Button>
</StackPanel>
```

La propietat *Orientation* del plafó indica l'orientació que prenen els components, que pot ser *Vertical* o *Horizontal*. La figura següent mostra el resultat del codi anterior amb els dos tipus d'orientació:

Figura 15. Exemple de plafó *StackPanel*

ScrollViewer

El control *ScrollViewer* no és un pròpiament un plafó, atès que es pot combinar amb un plafó per a afegir la funcionalitat de barres de desplaçament als components que conté.

El control *ScrollViewer* es pot utilitzar no solament amb plafons sinó també amb altres controls; per exemple, el codi següent fa servir un *ScrollViewer* per a fer *scroll* d'una imatge:

```
<ScrollViewer HorizontalScrollBarVisibility="Auto">
   <Image Source="flor_almendro.jpg"/>
</ScrollViewer>
```

7.1.5. Esdeveniments

En WPF tots els controls disposen d'un esdeveniment *Loaded*, que correspon a quan el control ha estat inicialitzat i està llest per a ser renderitzat. Un esdeveniment molt utilitzat és el *Window_Loaded*, ja que és l'esdeveniment que apareix per defecte en clicar sobre el fons d'una finestra a Visual Studio:

```
private void Window_Loaded(object o, RoutedEventArgs e)
{
}
```

A banda d'aquest esdeveniment, de manera molt similar a Windows Forms, els controls WPF disposen d'un ampli ventall d'esdeveniments: per exemple, en prémer el botó del ratolí, prémer una tecla, guanyar/perdre el focus, etc.

WPF incorpora com a novetat el protocol d'encaminament d'esdeveniments, que ofereix gran flexibilitat per a gestionar els esdeveniments de l'arbre visual d'elements que formen la interfície d'usuari.

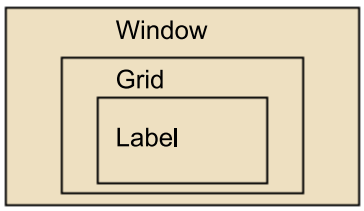
Tots els controls d'una finestra o pàgina WPF formen una estructura jeràrquica denominada *arbre visual*.

Arbre visual d'elements

A tall d'exemple, mostrem el codi XAML d'una pàgina, amb l'arbre visual corresponent:

```
<Window>
   <Grid>
      <Label>
      </Label>
   </Grid>
</Window>
```

Figura 16. Arbre visual d'elements



En WPF hi ha tres tipus d'esdeveniments:

- ***Direct***: és el funcionament clàssic que hi havia a Windows Forms, en què només es notifica a l'element en concret que va originar l'esdeveniment.

- ***Tunnel***: l'esdeveniment travessa l'arbre visual de dalt cap avall, començant per l'arrel, i acabant en l'element que va originar l'esdeveniment. Per convenció, aquests esdeveniments tenen el prefix *Preview*.

- ***Bubble***: l'esdeveniment travessa l'arbre visual de baix cap a dalt, començant en l'element que va originar l'esdeveniment, i acabant en l'arrel.

A l'exemple següent, mostrem els esdeveniments en funcionament. Per a això, associem una mateixa funció genèrica als esdeveniments *MouseDown* (de tipus *Bubble*) i *PreviewMouseDown* (de tipus *Tunnel*) dels elements *Window*, *Grid* i *Label*, i observarem en quin ordre s'executen les funcions:

- Codi de l'arxiu Esdeveniments.xaml:

```
<Window Name="window1"
   PreviewMouseDown="GenericRoutedEventHandler"
   MouseDown="GenericRoutedEventHandler">

   <Grid Name="grid1"
      PreviewMouseDown="GenericRoutedEventHandler"
      MouseDown="GenericRoutedEventHandler">

      <Label Content="Clica aquí" Name="label1"
      PreviewMouseDown="GenericRoutedEventHandler"
      MouseDown="GenericRoutedEventHandler">
      </Label>
   </Grid>
</Window>
```

- Codi de l'arxiu Esdeveniments.xaml.cs:

```
private void GenericRoutedEventHandler(object sender, RoutedEventArgs e)
{
  string name = ((FrameworkElement)sender).Name;
  Console.WriteLine(name + "\t" +
    e.RoutedEvent.Name + "\t" +
    e.RoutedEvent.RoutingStrategy);
}
```

En executar l'aplicació i prement el botó esquerre del ratolí sobre l'element *label1*, obtenim la sortida següent:

```
window1   PreviewMouseDown    Tunnel
grid1     PreviewMouseDown    Tunnel
```

```
label1    PreviewMouseDown    Tunnel
label1    MouseDown           Bubble
grid1     MouseDown           Bubble
window1   MouseDown           Bubble
```

En primer lloc, s'ha produït l'esdeveniment *PreviewMouseDown*, el qual, com que és de tipus *Tunnel*, ha començat el recorregut a l'arrel de l'arbre (la finestra), després *grid1* i en acabar *label1*. En canvi, l'esdeveniment *MouseDown* segueix l'ordre invers, ja que comença en l'element on s'ha originat l'esdeveniment (*label1*), i acaba a la finestra.

Val la pena comentar que l'exemple anterior ha estat dissenyat per a mostrar la propagació d'esdeveniments a WPF, però en les aplicacions WPF el més usual és parar la propagació de l'esdeveniment en qualsevol punt.

7.2. Controls

El tipus i la funció dels controls predefinits de WPF és molt similar al de WinForms, encara que els de WPF ofereixen molta més flexibilitat. Per exemple, els controls de WPF permeten afegir, com a contingut d'un control, altres controls. En realitat, cada control només pot contenir un únic control, però si aquest control és un plafó, dins seu s'hi poden organitzar nous controls, amb la qual cosa es multipliquen les combinacions possibles. Tot seguit, repassarem els controls principals de WPF:

Més controls

A part dels controls inclosos en el .NET Framework, podem afegir molts altres controls (n'hi ha tant de gratuïts com de comercials), o crear els nostres propis controls.

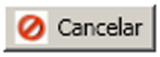
- ***Label***: mostra un text a l'usuari.
- ***TextBox***: és el control més comú d'introducció de dades.
- ***PasswordBox***: permet a l'usuari introduir contrasenyes de manera segura. La contrasenya introduïda per l'usuari es pot obtenir a través de la propietat *Password*, que retorna un objecte *SecureString* (és com un *String*, però s'emmagatzema xifrat en memòria).
- ***Button***: és un botó. El contingut no té per què ser exclusivament text, sinó que s'hi pot posar qualsevol tipus de contingut.

Exemple

A l'exemple següent, es fa servir un *StackPanel* per a agrupar una imatge i una etiqueta dins del botó:

```
<Button Name="button1" Height="23" Width="73">
   <StackPanel Orientation="Horizontal">
      <Image Source="cancelar.jpg" Width="24"/>
      <Label Content="Cancelar"/></StackPanel>
</Button>
```

Figura 17. Botó amb imatge i text



- ***CheckBox***: és un botó que pot estar marcat, o no, i obtenim aquest valor amb la propietat *IsChecked*.

- ***RadioButton***: és similar al *CheckButton*, però amb la diferència que mentre que diversos *CheckBox* poden estar activats alhora, d'entre tots els *RadioButton* que pertanyen a un mateix grup, només un pot estar activat cada vegada. En cas de no haver-hi cap grup, l'agrupació es faria entre tots els *RadioButton* d'un mateix plafó.

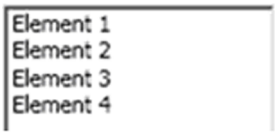
- ***RichTextBox***: és similar al *TextBox*, però més potent, ja que permet editar text amb format.

- ***ListBox***: mostra un conjunt d'elements en forma de llista.

Exemple de *ListBox*

```
<ListBox Name="listBox1">
  <ListBoxItem>Element 1</ListBoxItem>
  <ListBoxItem>Element 2</ListBoxItem>
  <ListBoxItem>Element 3</ListBoxItem>
  <ListBoxItem>Element 4</ListBoxItem>
</ListBox>
```

Figura 18. Control *ListBox*

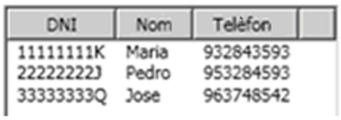


- ***ComboBox***: similar al *ListBox*, amb la diferència que és una llista desplegable, i l'usuari només pot seleccionar-hi un element.

- ***ListView***: permet mostrar dades d'una llista en diverses columnes.

Exemple de *ListView*

Figura 19. Control *ListView*

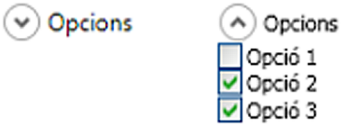


- ***Expander***: permet agrupar un conjunt de controls, i l'usuari el pot encongir o ampliar prement un botó.

Exemple d'*Expander*

```
<Expander Header="Opciones">
   <StackPanel>
      <CheckBox>Opción 1</CheckBox>
      <CheckBox>Opción 2</CheckBox>
      <CheckBox>Opción 3</CheckBox>
   </StackPanel>
</Expander>
```

Figura 20. Expander contret i expandit



- ***TabControl***: permet organitzar contingut en forma de pestanyes. Els elements d'un control *TabControl* són de tipus *TabItem*; cada un correspon amb una pestanya del control.

Exemple de *TabControl*

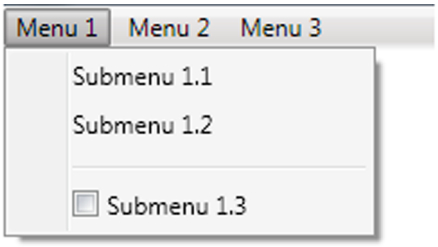
```
<TabControl>
  <TabItem Header="Tab 1">Aquest és el tab 1</TabItem>
  <TabItem Header="Tab 2">Aquest és el tab 2</TabItem>
  <TabItem Header="Tab 3">Aquest és el tab 3</TabItem>
</TabControl>
```

- ***Menu***: permet crear menús fàcilment. Dins del *Menu* s'afegeix un element *MenuItem* per cada menú que vulguem crear, i hi podem afegir una barra separadora amb un element *Separator*.

Exemple de *Menu*

```
<Menu Height="22" Name="menu1" VerticalAlignment="Top">
   <MenuItem Header="Menu 1">
      <MenuItem Header="Submenu 1.1"/>
      <MenuItem Header="Submenu 1.2"/>
      <Separator/>
      <CheckBox Content="Submenu 1.3"/>
   </MenuItem>
   <MenuItem Header="Menu 2"/>
   <MenuItem Header="Menu 3"/>
</Menu>
```

Figura 21. Control *Menu*

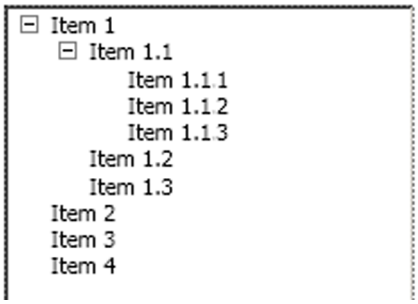


- ***ContextMenu***: tipus especial de menú que s'obre en prémer el botó secundari del ratolí damunt d'algun control.

- ***TreeView***: permet mostrar un conjunt d'elements de manera jeràrquica, és a dir, en forma d'arbre.

Exemple de *TreeView*

Figura 22. Control *TreeView*

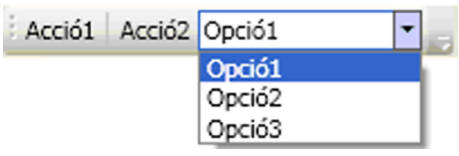


- ***ToolBar***: mostra una barra d'icones o eines, formada per botons, etiquetes o altres controls que vulguem afegir. El control *ToolBar* es pot afegir en qualsevol part de la interfície gràfica, encara que generalment s'inclou en un *ToolBarTray*, el qual pot contenir diverses barres d'eines.

Exemple de *ToolBar*

```
<ToolBarTray VerticalAlignment="Top">
<ToolBar Name="toolBar1">
    <Button>Accio1</Button>
<Separator/>
    <Button>Accio2</Button>
<ComboBox Width="100">
      <ComboBoxItem>Opcio1</ComboBoxItem>
      <ComboBoxItem>Opcio2</ComboBoxItem>
      <ComboBoxItem>Opcio3</ComboBoxItem>
</ComboBox>
</ToolBar>
</ToolBarTray>
```

Figura 23. Control *ToolBar*



- ***ProgressBar***: mostra una barra de progrés, útil, per exemple, per a mostrar el percentatge finalitzat d'una tasca determinada. Aquest control té les

propietats *Value*, *Minimum* i *Maximum*, que permeten establir o consultar el valor mostrat a la barra, el valor corresponent al mínim (barra buida), i el valor corresponent al màxim (barra completa), respectivament:

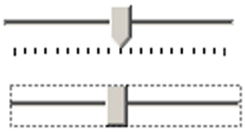
Exemple de *ProgressBar*

Figura 24. Control *ProgressBar*

- ***Slider***: permet que l'usuari seleccioni un valor determinat entre el mínim i màxim establerts. La propietat *TickPlacement* permet visualitzar o amagar les marques de medició.

Exemple de *Slider*

Figura 25. Control *Slider* amb marques de medició i sense



- ***WindowsFormsHost***: permet executar qualsevol control *WinForms* en una aplicació WPF. Per a fer-ho, n'hi haurà prou d'afegir a la finestra WPF un control *WindowsFormsHost*, i incloure-hi el control *WinForms* que s'hi vol executar.

7.3. Funcionalitats gràfiques i multimèdia

En els subapartats següents representem alguns exemples de funcionalitats gràfiques i multimèdia de WPF.

7.3.1. Transformacions

WPF permet fer transformacions geomètriques, com ara rotacions o translacions, dels diferents elements de la interfície. Algunes de les transformacions més comunes són les següents:

- ***RotateTransform***: permet fer girar un element de la interfície en un angle determinat.

Exemple de *RotateTransform*

L'exemple següent mostra com fer girar un botó 25 graus a la dreta:

```
<StackPanel Width="134">
   <Button Background="Red" Height="50" Width="100"> 1
   </Button>
   <Button Background="Orange" Height="50" Width="100"> 2
      <Button.RenderTransform>
         <RotateTransform Angle="25" />
```

```
        </Button.RenderTransform>
    </Button>
    <Button Background="Yellow" Width="100" Height="50"> 3
    </Button>
</StackPanel>
```

Figura 26. Rotació d'un botó

- ***TranslateTransform***: permet traslladar un element seguint un vector de translació determinat, indicat mitjançant les propietats *X* i *Y* de la classe *TranslateTransform*.

Exemple de *TranslateTransform*

Amb el codi següent, traslladem el botó central de l'exemple anterior:

```
<Button Background="Orange" Height="50" Width="100">
    <Button.RenderTransform>
        <TranslateTransform X="50" Y="-20" />
    </Button.RenderTransform>
    2
</Button>
```

Figura 27. Translació d'un botó

- ***ScaleTransform***: permet escalar un element en una proporció concreta determinada per les propietats següents:
  - *ScaleX*: factor d'escalat en horitzontal
  - *ScaleY*: factor d'escalat en vertical
  - *CenterX*: coordenada X del centre d'escalat
  - *CenterY*: coordenada Y del centre d'escalat

Exemple

L'exemple següent fa un escalat del botó central amb factors d'escala d'1,5 × 2, i centre d'escalat (50,0):

```
<Button Background="Orange" Height="50" Width="100">
<Button.RenderTransform>
<ScaleTransform ScaleX="1.5" ScaleY="2" CenterX="50"/>
</Button.RenderTransform>
  2
</Button>
```

Com podem comprovar en el resultat, l'ampliació horitzontal es fa de manera uniforme cap als dos costats, perquè la coordenada *x* del centre d'escalat se situa enmig del botó, mentre que l'escalat vertical es fa només cap avall perquè la coordenada *i* del centre d'escalat s'estableix a zero per defecte.

Figura 28. Aplicació de *ScaleTransform*

- ***TransformGroup***: permet combinar múltiples transformacions sobre un element, que tenen lloc segons l'ordre en el qual s'han afegit al *TransformGroup*.

Exemple de *TransformGroup*

L'exemple següent aplica una rotació i, a continuació, una translació:

```
<Button Background="Orange" Height="50" Width="100">
   <Button.RenderTransform>
      <TransformGroup>
         <RotateTransform Angle="45"/>
         <TranslateTransform X="50" Y="-50"/>
      </TransformGroup>
   </Button.RenderTransform>
   2
</Button>
```

Figura 29. Resultat de la combinació de rotació i translació

7.3.2. Animacions

WPF incorpora diverses classes que permeten crear animacions, entenent per *animació* la modificació d'una propietat durant un determinat interval de temps i entre un determinat rang de valors.

Les classes es denominen *XXXAnimation*, en què *XXX* és el tipus de dada de la propietat que s'animarà. WPF suporta els tipus de dades *Char*, *Color*, *Int16*, *Int32*, *Int64*, *Double*, etc.

Exemple

El codi següent fa que el botó canviï la seva amplada des de 50 a 200 en 2 segons:

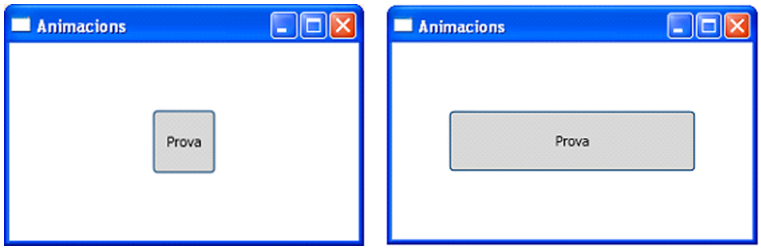
- Codi de l'arxiu Window1.xaml

```
<Button Name="boton1" Width="50" Height="50">
   Prova
</Button>
```

- Codi de l'arxiu Window1.xaml.cs

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
   DoubleAnimation da = new DoubleAnimation();
   da.From = 50;
   da.To = 200;
   da.Duration = new Duration(new TimeSpan(0, 0, 2));
   boton1.BeginAnimation(Button.WidthProperty, da);
}
```

Figura 30. Inici i final de l'animació



Rotació d'una imatge

En l'exemple següent fem la rotació d'una imatge:

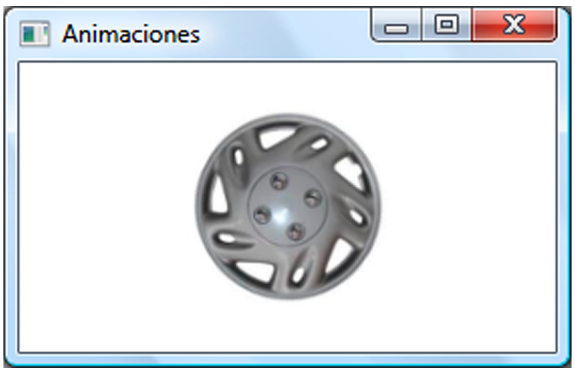
- Codi de l'arxiu Window1.xaml

```
<Image Name="imagen" Source="imagen.jpg" Width="100">
</Image>
```

- Codi de l'arxiu Window1.xaml.cs

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
   DoubleAnimation da = new DoubleAnimation();
   da.From = 0;
   da.To = 360;
   da.Duration = new Duration(new TimeSpan(0, 0, 2));
   da.RepeatBehavior = RepeatBehavior.Forever;
   RotateTransform rt = new RotateTransform();
   imagen.RenderTransform = rt;
   imagen.RenderTransformOrigin = new Point(0.5, 0.5);
   rt.BeginAnimation(RotateTransform.AngleProperty, da);
}
```

Figura 31. Imatge en moviment



7.3.3. Àudio i vídeo

En WPF, reproduir un arxiu d'àudio és tan fàcil com incloure el fragment següent de codi a l'aplicació:

```
System.Media.SoundPlayer sp;
sp = new System.Media.SoundPlayer("sonido.wav");
sp.Play();
```

I per a reproduir un arxiu de vídeo, simplement hem d'afegir un control de tipus *MediaElement*, al qual li indiquem en la propietat *Source* el vídeo que volem reproduir. Una vegada fet això, amb els mètodes *Play*, *Pause* i *Stop* de l'objecte podrem controlar la reproducció:

```
<MediaElement Name="video1" Source="video.wmv" />
```

7.4. WPF Browser i Silverlight

En aquest subapartat, tractarem les dues tecnologies que permeten crear aplicacions web mitjançant WPF, que són WPF Browser i Silverlight.

7.4.1. WPF Browser

Les aplicacions WPF Browser s'executen des d'un navegador, i tenen accés a totes les característiques gràfiques i multimèdia de WPF, per la qual cosa requereixen que el client tingui instal·lat el .NET Framework 3.0. De fet, en una aplicació WPF Browser podem fer gairebé el mateix que en una aplicació WPF d'escriptori. Una de les diferències principals és que les aplicacions WPF Browser s'executen dins d'uns estrictes límits de seguretat, per la qual cosa no tenen accés ni al registre, ni al sistema de fitxers.

Els URL de les aplicacions WPF Browser corresponen a un arxiu amb extensió .xbap i, quan un navegador hi accedeix, l'aplicació sencera es descarrega i desa temporalment a la memòria cau del navegador. Una vegada descarregada en local, el navegador executa automàticament l'aplicació, ja que no li cal cap procés d'instal·lació.

Per a desenvolupar una aplicació WPF Browser amb Visual Studio 2008, cal crear un projecte de tipus WPF Browser Application. Un cop creat, ja podem programar l'aplicació igual com faríem per a WPF, només amb la diferència que, en executar-la, es llança automàticament un navegador que apunta a l'arxiu .xbap corresponent a l'aplicació compilada.

7.4.2. Silverlight

Silverlight també permet crear aplicacions web mitjançant WPF, però, a diferència de WPF Browser, Silverlight és únicament un subconjunt de WPF, per la qual cosa les seves funcionalitats són molt més limitades.

Més sobre Silverlight

Per a més informació, podeu accedir a la pàgina web de Silverlight.

Per tal de desenvolupar una aplicació Silverlight amb Visual Studio 2008, és necessari instal·lar prèviament el paquet denominat Silverlight Tools for Visual Studio 2008. Una vegada instal·lat, en crear un projecte de tipus Silverlight Application, automàticament es crea una solució amb dos projectes: un projecte Silverlight, on farem tota la implementació, i un projecte ASP.NET, que té una pàgina HTML amb la referència a l'arxiu .xap de l'aplicació Silverlight.

Dos projectes

El motiu que fa que es creï un projecte Silverlight i un projecte ASP.NET és que, a diferència de les aplicacions WPF Browser, les aplicacions Silverlight han de ser allotjades en una pàgina HTML inicial (igual com passa, per exemple, amb les *applets* de Java).

8. Windows Mobile

Aquest apartat ofereix una introducció al desenvolupament d'aplicacions per a dispositius mòbils, i concretament per a Windows Mobile.

8.1. Una introducció a Windows Mobile

En aquest subapartat veurem quins dispositius permeten executar aplicacions .NET, i també els seus sistemes operatius i eines de desenvolupament disponibles.

8.1.1. Dispositius

Pocket PC (PPC, abreujat) és una especificació maquinari per a ordinadors de butxaca que incorporen el sistema operatiu Windows Mobile de Microsoft. A més, aquests dispositius poden treballar amb múltiples aparells afegits, com, per exemple, receptors de GPS, lectors de codis de barres, càmeres, etc.

Els dispositius amb telèfon i pantalla no tàctil es diuen *smartphones*. La diferència principal és que un *smartphone* està pensat per a ser un telèfon mòbil i fer-se servir amb una mà, mentre que un *pocket PC* té la funció d'agenda electrònica i es pot usar amb totes dues mans. De tota manera, tendeixen a unificar-se, amb la qual cosa cada vegada és menys clara la diferència entre tots dos.

Figura 32. Exemples de *Pocket PC* i *smartphones*.

Per a sincronitzar dades i comunicar-se, aquests dispositius incorporen mecanismes com ara USB, WiFi, Bluetooth o IrDa (infraroigs). La sincronització de les dades amb un PC es duu a terme amb el programa ActiveSync de Microsoft o mitjançant el centre de sincronització de dispositius a Windows Vista.

Actualment, hi ha milers d'aplicacions comercials per a *pocket PC*, i també podem desenvolupar les nostres pròpies aplicacions amb C++ Embedded (per a obtenir el màxim rendiment), o de manera més fàcil amb .NET.

8.1.2. Sistemes operatius

Dispositius amb Windows CE

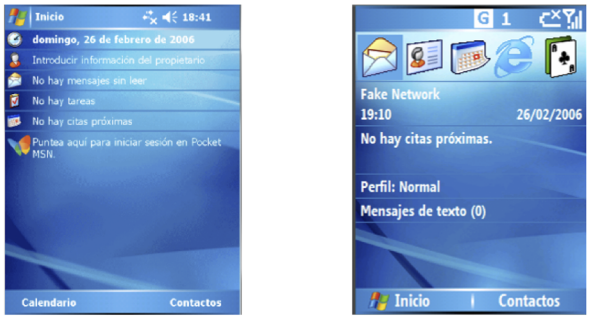
Un exemple de dispositius amb Windows CE són els navegadors Tom Tom ONE.

Windows CE és un sistema operatiu modular dissenyat per a dispositius encastats, i amb una capacitat de procés i emmagatzemament limitada. Com que és modular, Windows CE s'ajusta a les necessitats de cada dispositiu, i selecciona i configura únicament els components que volem. A més, els fabricants acostumen a incloure les seves pròpies aplicacions dins d'aquest sistema.

Hi ha fabricants de dispositius, com, per exemple, Symbol o Intermec, que en comprar els dispositius ens permeten escollir entre Windows CE i Windows Mobile. Com a usuaris finals, podem detectar fàcilment la diferència, ja que en Windows CE el botó d'inici acostuma a ser a la cantonada inferior esquerra (com a Windows), mentre que en Windows Mobile l'inici està a la cantonada superior esquerra. D'altra banda, en Windows CE hi ha barra de tasques, i en Windows Mobile no n'hi ha.

El sistema operatiu utilitzat els *pocket PC* i els *smartphones* és Windows Mobile, que és un sistema operatiu basat en Windows CE.

Figura 33. Pantalla d'inici de Windows Mobile per a *pocket PC* i *smartphones*



A la llista següent, enumerem les versions més recents de Windows Mobile, així com la versió de Windows CE en què estan basades:

- Windows Mobile 2003 (basat en Windows CE 4.2)
- Windows Mobile 2003 Second Edition (basat en Windows CE 4.2)
- Windows Mobile 5 (basat en Windows CE 5.0)
- Windows Mobile 6 (basat en Windows CE 5.2)

Concretament, l'últim Windows Mobile té diferents edicions:

- **Windows Mobile Classic** (Pocket PC): per a dispositius sense telèfon i amb pantalla tàctil.
- **Windows Mobile Standard** (Smartphone): per a dispositius amb telèfon i sense pantalla tàctil.

- **Windows Mobile Professional** (Pocket PC Phone Edition): per a dispositius amb telèfon i pantalla tàctil.

8.1.3. Eines de desenvolupament

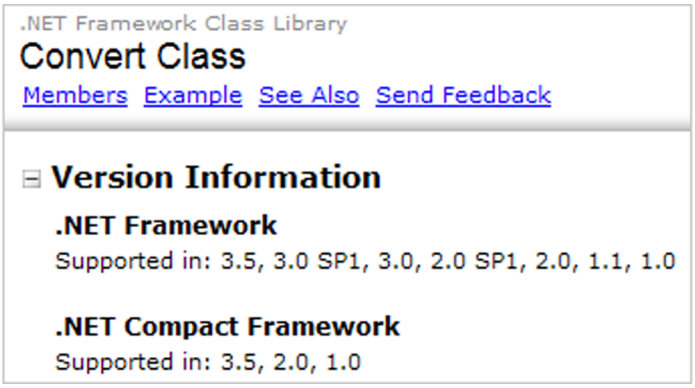
El desenvolupament d'aplicacions .NET per a Windows Mobile es fa amb una versió més limitada del *framework*, anomenada .NET Compact Framework (.NET CF), que té un nombre inferior de classes i mètodes. El .NET Framework ocupa uns 40 MB de memòria, que resulta ser massa per als *pocket PC*, que habitualment tenen 64 o 128 MB de RAM. El .NET CF permet reduir l'espai requerit a uns 4 MB de RAM, i implementa aproximadament un 30% de les classes del .NET framework.

En la documentació del .NET Framework s'identifiquen les classes i membres disponibles per al .NET CF, així com les versions concretes en les quals estan disponibles.

Documentació de la classe *Convert*

A l'exemple següent, mostrem la documentació de la classe *Convert*:

Figura 34. Documentació del .NET Framework en què s'indica la compatibilitat amb .NET CF



Visual Studio facilita el desenvolupament d'aplicacions de .NET CF, ja que proporciona diferents tipus de projecte i dissenyadors específics per a aquesta plataforma. També permet la depuració i execució d'aplicacions de .NET CF de manera integrada, amb els emuladors de cada plataforma.

Figura 35. Emuladors de Windows Mobile per a *pocket PC* i *smartphone*

8.2. WinForms per a dispositius mòbils

El desenvolupament d'aplicacions WinForms per a dispositius mòbils està optimitzat per a millorar el rendiment i reduir-ne la mida. Per aquest motiu, s'han eliminat múltiples funcionalitats poc usables per a dispositius mòbils, com ara *drag&drop*, impressió, controls ActiveX, etc.

8.2.1. Primera aplicació amb .NET CF

Per tal de crear una aplicació per a dispositius mòbils usant .NET CF, crearem un projecte de tipus *Smart Device* a Visual Studio. A continuació, seleccionarem la plataforma de destinació (*pocket PC*, *smartphone*, etc.), la versió de .NET CF, i el tipus de projecte.

Atenció!

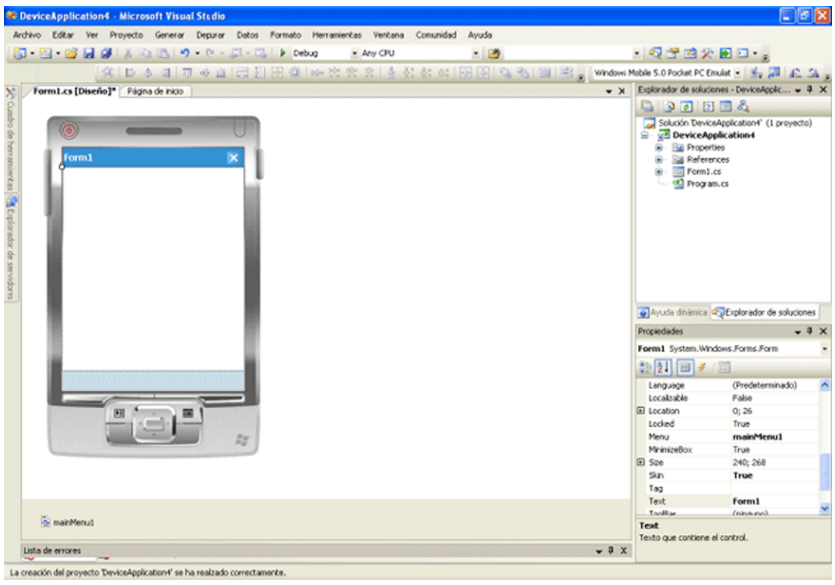
En cas que no aparegui cap plataforma de Windows Mobile 6, haurem de descarregar el Windows Mobile 6 SDK i instal·lar-lo.

Visual Studio afegeix automàticament diversos fitxers al projecte, concretament Program.cs (programa principal), i Form1.cs (un formulari). Immediatament després, s'obre el dissenyador de formularis específic per a la plataforma que hàgim seleccionat, on podrem arrossegar els controls i anar afegint el codi corresponent als diferents esdeveniments.

Projecte per a *pocket PC*

A l'exemple següent, hem creat un projecte per a *pocket PC*.

Figura 36. Dissenyador de WinForms per a dispositius mòbils



8.2.2. Formularis WinForms

La interacció amb els formularis és diferent segons si es tracta d'un dispositiu *pocket PC* o un *smartphone*:

1) Pocket PC

Els dispositius *pocket PC* permeten interactuar amb les aplicacions mitjançant un llapis digital utilitzat com si fos un ratolí, i l'entrada de dades es produeix amb un teclat que apareix en pantalla o utilitzant un component de reconeixement de caràcters.

A la part superior de la finestra, hi ha una barra d'estat amb el títol de la finestra i una sèrie d'icones de notificació, que indiquen paràmetres com el volum, l'hora o la quantitat de bateria disponible. A la dreta del tot, hi ha una icona que permet ocultar la finestra. Aquesta icona és configurable amb la propietat *MinimizeBox* del formulari, i pot ser *x* o *ok*. En el primer cas, en prémer el botó minimitzem la finestra, però l'aplicació es continua executant en segon pla. En el segon cas tanquem la finestra, i finalitzem l'execució de l'aplicació.

2) SmartPhone

Els dispositius *smartphone* normalment no disposen de pantalla tàctil, per la qual cosa la interacció amb l'usuari es fa mitjançant els botons del telèfon. A la part superior de la finestra, apareix també una barra d'informació amb el títol de la finestra i algunes icones que indiquen, per exemple, l'estat de la bateria i la cobertura. Tanmateix, aquesta barra d'informació no té icona per tancar la finestra, la qual pot tancar-se prement la tecla d'inici o la tecla de cancel·lar,

tot i que només es minimitza perquè continua executant-se en segon pla. Per a tancar l'aplicació, cal afegir una opció addicional al programa, com ara una opció de menú.

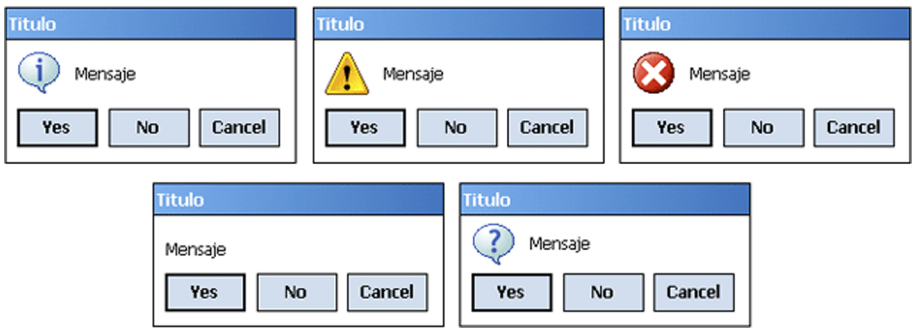
8.2.3. Quadres de diàleg

Els quadres de diàleg són també formularis, tot i que es fan servir puntualment per a mostrar un missatge a l'usuari o demanar-li que seleccioni o introdueixi una dada determinada. Dins del *namespace* de WinForms, hi ha alguns quadres de diàleg predefinits que permeten fer tasques comunes.

Quadres de diàleg predefinits	
CommonDialog	És una classe base a partir de la qual podem crear altres quadres de diàleg.
FileDialog	Mostra un quadre de diàleg en el qual l'usuari pot seleccionar un arxiu. És una classe abstracta que serveix de base per a les classes *OpenFileDialog* i *SaveFileDialog*.
OpenFileDialog	Mostra un quadre de diàleg en el qual l'usuari pot seleccionar un arxiu per a obrir-lo.
SaveFileDialog	Mostra un quadre de diàleg en el qual l'usuari pot seleccionar un arxiu per a desar-lo.
MessageBox	Mostra un missatge a l'usuari i permet seleccionar una acció per a executar-la.

L'aspecte del quadre de diàleg *MessageBox* a *pocket PC* té l'aspecte que mostrem a la figura següent. Les icones mostrades corresponen als valors de l'enumeració *MessageBoxIcon* en ordre (*Asterisk*, *Exclamation*, *Hand*, *None* i *Question*):

Figura 37. Quadres de diàleg *MessageBox* a Pocket PC



8.2.4. Controls del .NET CF

La creació d'una aplicació WinForms per a dispositius mòbils es fa de la mateixa manera que en WinForms normal, però amb un nombre de controls disponibles inferior. A més, molts dels controls només estan disponibles per als *pocket PC*, ja que els *smartphones*, com que no tenen pantalla tàctil, disposen d'un reduït nombre de controls.

A la taula següent, mostrem els controls principals i en quins tipus de dispositius estan disponibles:

Control	*Pocket PC*	*Smartphone*
Button	Disponible	No disponible
CheckBox	Disponible	Disponible
ComboBox	Disponible	Disponible
ContextMenu	Disponible	No disponible
DataGrid	Disponible	Disponible
DateTimePicker	Disponible	Disponible
HScrollBar	Disponible	Disponible
Label	Disponible	Disponible
ListBox	Disponible	No disponible
ListView	Disponible	Disponible
MainMenu	Disponible	Disponible
Panel	Disponible	Disponible
PictureBox	Disponible	Disponible
ProgressBar	Disponible	Disponible
RadioButton	Disponible	No disponible
StatusBar	Disponible	No disponible
TabControl	Disponible	No disponible
TextBox	Disponible	Disponible
Timer	Disponible	Disponible
ToolBar	Disponible	No disponible
TreeView	Disponible	Disponible
VScrollBar	Disponible	Disponible

En dissenyar una aplicació per a Windows Mobile, és important tenir en compte una sèrie de normes, que estan descrites en el programa de Microsoft "Designed for Windows Mobile". A continuació, esmentem algunes d'aquestes normes:

- No utilitzar fonts de lletra de mida fixa.

- Disposar d'un únic nivell de menús (sense submenús).

- Si l'aplicació està dissenyada per a ser utilitzada amb el Stylus (el "llapis" de la PDA), els botons han de ser de 21 × 21 píxels. En canvi, si s'han de prémer amb el dit, la mesura ha de ser de 38 × 38 píxels.

- Revisar que tot es vegi bé, independentment de la il·luminació.

- Col·locar els elements que s'hagin de prémer, a la part baixa de la pantalla. D'aquesta manera, la mà de l'usuari no taparà la pantalla.

- Reduir al màxim les dades que l'usuari hagi d'escriure mitjançant el teclat virtual (anomenat SIP, de l'anglès *Software Input Panel*).

- No omplir les llistes amb un nombre gaire elevat d'elements.

- Totes les aplicacions han de registrar una icona de 16 × 16 píxels, i un altre de 32 × 32 píxels.

- La barra de navegació ha de mostrar sempre el nom de l'aplicació, i cap altra dada més.

- Tota aplicació ha de disposar d'un instal·lador que s'encarregui de crear un accés directe a l'executable. A més, haurà de disposar d'un desinstal·lador que ho deixi tot tal com estava inicialment.

8.2.5. Desplegament d'aplicacions

En executar una aplicació, Visual Studio la copia automàticament al dispositiu o a l'emulador. Si aquest no té instal·lada una versió compatible del .NET CF, Visual Studio la instal·la automàticament.

Recomanació

Si podem, és més recomanable treballar amb un dispositiu real que amb l'emulador.

L'altra possibilitat és fer una instal·lació manual, que consisteix a copiar els assemblatges de l'aplicació en una carpeta del dispositiu.

Si volem distribuir l'aplicació, serà necessari oferir un programa d'instal·lació. Per a això, crearem un projecte de tipus *Other project types/Setup and deployment/ CAB Project*, el qual acabarà generant un arxiu CAB que instal·larà l'aplicació de manera molt senzilla.

8.3. Aplicacions web per a dispositius mòbils

El desenvolupament d'aplicacions web per a dispositius mòbils és més complex que per a webs normals per diversos motius:

- Hi ha diferents llenguatges suportats per diversos dispositius, per exemple, HTML, WML o cHTML.

cHTML

cHTML (Compact HTML) és un subconjunt d'HTML utilitzat als telèfons mòbils DoCoMo del Japó.

- Els dispositius poden tenir pantalles molt variades: diferents mides, diferents nombres de files i columnes de text, orientació horitzontal o vertical, pantalles en color o en blanc i negre, etc.

- Les velocitats de connexió poden ser molt diferents entre dispositius: GPRS (40 kbps), EDGE (473 kbps) o HSDPA (14 Mbps).

Per a resoldre aquestes diferències, Visual Studio 2005 oferia suport al desenvolupament d'aplicacions ASP.NET específiques per a dispositius mòbils, mitjançant els controls ASP.NET mobile. Quan un dispositiu es connecta a una pàgina ASP.NET mobile, el servidor recupera informació sobre el maquinari del dispositiu, el navegador i la velocitat de connexió. A partir d'aquesta informació, els controls ASP.NET mobile produeixen una sortida o una altra en funció del llenguatge de marcatge utilitzat, les capacitats del navegador, les propietats de la pantalla i la velocitat de connexió.

Atenció!

Visual Studio 2008 actualment no ofereix suport específic per al desenvolupament d'aplicacions ASP.NET per a mòbils.

D'aquesta manera, a Visual Studio 2005, per tal de crear una aplicació amb pàgines ASP.NET per a mòbils, cal crear un lloc web ASP.NET i afegir-hi formularis Mobile Web Forms. Un cop fet això, podrem comprovar com la pàgina ASP.NET hereta de MobilePage, i la paleta de controls mostra únicament els controls disponibles per a mòbils.

Recentment, els dispositius mòbils han evolucionat molt, fins al punt que, avui en dia, hi ha diversos models que permeten navegar sense problemes per pàgines HTML "normals". En conseqüència, i a causa del gran esforç que representa el desenvolupament dels controls ASP.NET Mobile, sembla probable pensar que Microsoft acabi abandonant aquesta línia de treball, i se centri en les pàgines ASP.NET "normals", que en breu seran accessibles per la majoria de dispositius mòbils del mercat.

Esforç de desenvolupament

Cal assegurar la compatibilitat amb infinitat de mòbils diferents, cosa que fa el desenvolupament molt carregós.

D'altra banda, val la pena comentar que Silverlight ja està disponible per a Symbian S60, i en breu estarà disponible per a Windows Mobile.

Bibliografia

Gañán, D. (2008). Material del màster de .NET de la UOC.

Gibbs, M.; Wahlin, D. (2007). *Professional ASP.NET 2.0 AJAX*. Wiley Publishing. Inc.

Johnson, G.; Northrup, T. (2006). *Microsoft .NET Framework 2.0 Web-Based Client Development*. Microsoft Press.

Johnson, B.; Madziak, P.; Morgan, S. (2008). *NET Framework 3.5 Windows Communication Foundation*. Microsoft Press.

Northrup, T.; Wildermuth, S. (2008). *Microsoft .NET Framework Application Development Foundation* (2a. ed.). Microsoft Press.

Stoecker, M. A.; Stein, S. J.; Northrup, T. (2006). *Microsoft .NET Framework 2.0 Windows-Based Client Development*. Microsoft Press.

Stoecker, M. A. (2008). *Microsoft .NET Framework 3.5 Windows Presentation Foundation*. Microsoft Press.

Wigley, A.; Moth, D.; Foot, P. (2007). *Microsoft Mobile Development Handbook*. Microsoft Press.