

Sviluppo



Di Michele Costabile



Quali sono i plus della soluzione di Microsoft per creare interfacce utente ed effetti grafici.

L'equivalente Xaml di questo codice sarebbe qualcosa simile a questo:

```
<Button Width="258" Height="40"
Margin="30,30,0,0" Content="Click
me" Click="Button_Click"
VerticalAlignment="Top" />
```

Introduzione a Xaml

Xaml è un linguaggio per la descrizione di interfacce utente ed effetti grafici, su cui Microsoft ha iniziato a lavorare nel 2003, dandogli il nome in codice Avalon. Faceva parte di Longhorn, ovvero il futuro Windows Vista, ancora in fase di sviluppo. Xaml venne ufficialmente rilasciato nel 2006 e l'anno dopo divenne una tecnologia alla base di Silverlight, l'edizione scalata di .net con cui Microsoft si propose di sottrarre quote di mercato a Flash. In tutti questi anni, Xaml è stato puntualmente aggiornato a ogni release del framework .net. La versione corrente è la 4.5. Chi fosse interessato all'intera timeline, può trovarne un riassunto all'indirizzo japf.github.io/xaml-history. Da più di dieci anni Microsoft investe su una macchina virtuale con il suo framework (.net) e su un linguaggio per la descrizione dichiarativa di interfacce utente e animazioni (xaml). La macchina virtuale ha portato nel mondo Microsoft il nucleo di due punti di forza di Java: indipendenza dall'hardware e possibilità di controllare il codice in esecuzione, per identificare precocemente codice maligno. L'altro membro della coppia, Xaml, ha portato indipendenza da interfacce applicative, compatibilità binaria e linking dinamico per la creazione del componente più critico per il successo di una piattaforma software, la parte visibile. Entrambi i membri dell'equazione hanno dato a Microsoft la possibilità di unificare la versione del sistema per telefoni e tablet con la piattaforma principale. Microsoft

è riuscita a passare con disinvoltura da un'architettura Intel a una basata su Arm e a realizzare una solida piattaforma per telefoni e tablet, grazie a .net e Xaml. Da Silverlight è arrivato Blend uno strumento nato per creare animazioni, che si integra perfettamente con Visual Studio condividendo i progetti. Quindi, sviluppando con Xaml abbiamo la possibilità di dare a designer e sviluppatori due strumenti differenti, progettati per le esigenze specifiche di utenti diversi, che condividono il medesimo progetto. Xaml è arrivato più lontano di Windows Forms, che viene conservato per compatibilità, soprattutto per la flessibilità di una descrizione dichiarativa, rispetto a un algoritmo espresso in codice, esattamente come siamo abituati a fare sul web. Codice come quello che segue, infatti, è rigido e non ha la possibilità di migrare facilmente da un framework a un altro, da un processore all'altro, da un set di regole di stile a un altro.

```
public Button button1;
public Form1()
{
    button1 = new Button();
    button1.Size = new Size(40,
40);
    button1.Location = new
Point(30, 30);
    button1.Text = "Click me";
    this.Controls.Add(button1);
    button1.Click += new
EventHandler(button1_Click);
}
```

Il modello applicativo di Xaml è il seguente: il codice xml che descrive l'interfaccia viene caricato da una funzione del framework, che provvede a istanziare i controlli nel codice seguendo la descrizione dell'interfaccia. Naturalmente, dato che il formato xml è dichiarativo, il modo in cui sono creati i controlli dal codice e l'ordine delle istruzioni non sono espliciti, ma saranno calcolati dal codice di supporto. Qui sta la chiave della separazione fra la descrizione dell'interfaccia e la sua costruzione. Ulteriori punti a favore di Xaml, sono la possibilità di creare stili per gli oggetti di interfaccia, in modo simile a quello che ha reso vincente la coppia html e css. La stilizzazione permette, per esempio, di cambiare sfondi e dimensione dei bordi in un'intera applicazione consistentemente, agendo su un parametro globale invece che su ogni singolo elemento di interfaccia. Questo semplifica la personalizzazione per diversi tipi di schermo e di formati. Vedremo in seguito che ci sono meccanismi potenti per specificare in modo dichiarativo le relazioni fra il contenuto degli elementi di interfaccia e la loro rappresentazione nel codice, in modo che si possano specificare relazioni, per esempio fra campi di testo o etichette di pulsanti e variabili di programma. Un'altra funzione utile di Xaml consente di creare *trigger*, cioè situazioni che causano mutamenti. Per esempio, valori di un campo di testo che abilitano o disabilitano una checkbox. Anche questo codice viene prodotto dal framework e scatta nel momento giusto, senza programmazione

esplicita. Insomma, Xaml è la tecnologia su cui investire per i nuovi progetti basati su Windows.

STRUTTURA DI XAML

Abbiamo detto che Xaml è un dialetto xml, quindi la definizione di un'interfaccia utente avviene in un file di testo, con la familiare struttura e verbosità, di una struttura xml. odice ospite di una finestra vuota nello schema di un'applicazione Windows universale, cioè pronta per andare su pc, tablet e telefono.

```
<Page
  x:Class="App1.MainPage"
  xmlns="http://schemas.
microsoft.com/winfx/2006/xaml/
presentation"
  xmlns:x="http://schemas.
microsoft.com/winfx/2006/xaml"
  xmlns:local="using:App1"
  xmlns:d="http://schemas.
microsoft.com/expression/
blend/2008"
  xmlns:mc="http://
schemas.openxmlformats.org/
markup-compatibility/2006"
  mc:Ignorable="d">
```

```
<Grid Background="{ThemeReso
urceApplicationPageBackgroundThem
eBrush}">
```

```
</Grid>
</Page>
```

Nulla di notevole in questo esempio. Le prime righe sono la consueta giaculatoria di namespace a cui fa riferimento la definizione formale del linguaggio. Il contenitore centrale, la cornice in cui inserire il resto dell'applicazione è una Grid, cioè una struttura in cui gli oggetti sono disposti per righe e colonne. La Grid specifica uno sfondo, che a sua volta fa riferimento a un tema. Abbiamo già trovato qualcosa di interessante. Premendo F12 andiamo a trovare la definizione di questo tema in un file di risorse.

```
<SolidColorBrush x:Key="Applic
ationPageBackgroundThemeBrush"
Color="#FF1D1D1D" />
```

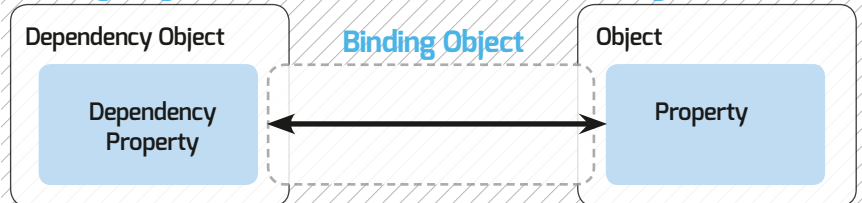
Se facciamo la stessa cosa nella versione mobile del template, seguendo la catena di definizioni, arriviamo a un colore diverso

```
<Color x:Key="PhoneBackgroundCo1o
```

CREARE UN LEGAME TRA INTERFACCIA E OGGETTI

Binding Target

Binding Source

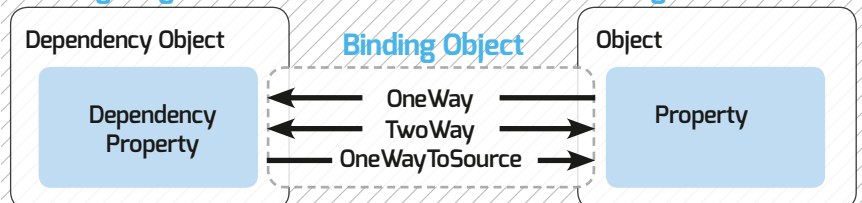


Il binding stabilisce un collegamento fra un elemento dell'interfaccia, per esempio un campo di testo, e una variabile di programma, come un campo estratto da un database

SCAMBIARE INFORMAZIONI ATTRAVERSO IL BINDING

Binding Target

Binding Source



Il binding supporta diverse direzioni, consentendo l'inizializzazione dell'interfaccia, o la valorizzazione delle variabili di programma con gli input dell'utente

```
r">#FF000000</Color>
```

Ecco, fin dalla prima riga di codice, l'esemplificazione di aspetto da mettere in risalto di Xaml: la flessibilità e la portabilità del codice. Gli stili permettono facilmente di fare cambiamenti globali toccando poche righe di stile generale. Parlando della struttura del documento, notiamo, di passaggio, che Xaml consente due stili di programmazione, uno basato su attributi e uno basato su elementi. Il secondo, è molto prolisso e viene evitato nel codice generato da Visual Studio, ma comunque è codice legale, e si può incontrare in qualche caso.

Ecco un esempio del primo stile di codice:

```
<Button Content="Button"
HorizontalAlignment="Left"
Height="122" Margin="73,154,0,0"
VerticalAlignment="Top"
Width="363"/>
```

Questo, invece è il secondo.

```
<Button>
  <Button.Content>
```

```
Button
  </Button.Content>
  <Button.
HorizontalAlignment>
    Left
  </Button.
HorizontalAlignment>
  ...
</Button>
```

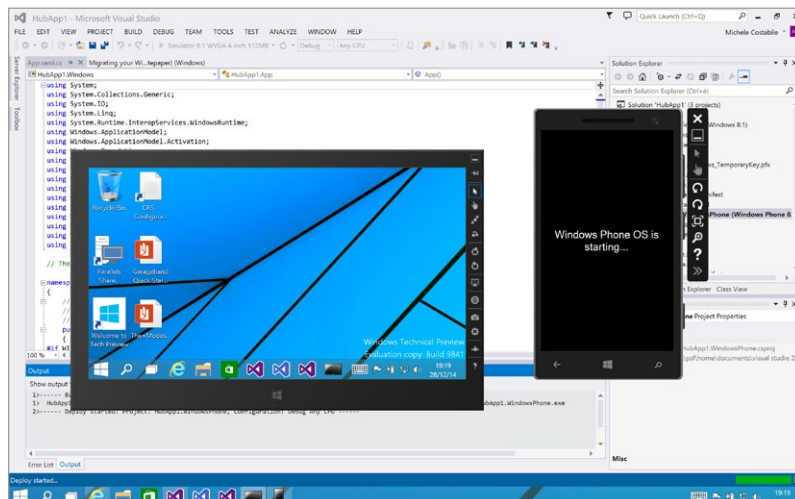
DATA BINDING

Il lavoro più pesante che fanno le applicazioni, specie quelle per uso gestionale, è prendere i dati da qualche parte, mostrarli sul video in modo comprensibile, accettare gli input dell'utente, validare con scrupolosa e maniacale precisione i dati inseriti e trasferirli in qualche struttura di memorizzazione permanente. Lungo la strada, spesso i dati sono trasformati in vario modo.

Chi scrive applicazioni, sa che passerà quattro quinti del tempo di sviluppo a occuparsi dell'interfaccia utente e di tutti i problemi di trasformazione e validazione dei dati. I problemi tecnologici di un'applicazione, per esempio

la connessione con servizi remoti o il pilotaggio di dispositivi particolari, come lettori di carte di credito, sono di norma più facili da risolvere, anche se richiedono studio intenso e la soluzione di problemi di elevata complessità. Sono, infatti, problemi in cui la complessità è elevata e localizzata, che si risolvono in fasi aperte e chiuse nello spazio di una settimana. Non stupisce, allora, che Xaml sia dotato di una ricca selezione di strumenti per regolare il passaggio dei dati in modo automatico e ordinato fra le variabili di programma e l'interfaccia utente. La chiave del meccanismo, sta nel concetto di binding, cioè il legame, fra una variabile di programma e qualche elemento di interfaccia utente. Il binding viene specificato in modo dichiarativo nel file Xaml e soddisfatto a run time. Questo vuol dire che il sistema riesce a creare un legame fra una variabile e una proprietà di un elemento di interfaccia in modo dinamico.

Si tratta di un meccanismo facile da usare, ma che pone problemi tecnologici non lievi al framework. Per esempio, nel file Xaml viene specificato il nome di una variabile, che deve essere risolto con un link dinamico a run time usando la reflection, cioè la capacità del codice di convertire il nome di una variabile in un indirizzo di programma durante l'esecuzione, sfruttando le tabelle create dal linker e conservate nella struttura dell'eseguibile. Quando il sistema ha individuato la variabile collegata allo stato di un elemento di interfaccia grazie alla reflection deve collegarla con l'elemento richiesto, per esempio il valore di un campo di testo o lo stato di una checkbox. Durante il percorso può anche essere necessaria una conversione, per esempio fra un valore e un colore. Tutti questi problemi sono risolti dalla soluzione Microsoft, in modo che accada proprio quello che il programmatore si aspetta nelle diverse



Un'applicazione universale aperta nella versione gratuita di Visual Studio 2013. I modelli di app universale sono progettati con la segmentazione del codice che consente di creare contemporaneamente app per tutte le piattaforme

situazioni. Per esempio si può inserire un ValueConverter sul percorso di un binding. Il Data Binding può avvenire in diverse direzioni, in accordo con diverse esigenze applicative. Il trasporto dei dati può avvenire dallo stato del codice verso l'interfaccia, da questa verso i modelli applicativi, o in entrambe le direzioni, per esempio in una transazione di modifica.

Dal punto di vista di un modello di programmazione tradizionale, per esempio Visual Basic, è come avere la possibilità di creare automaticamente gestori dei dati da inserire nel caricamento dell'interfaccia (Form.OnLoad) e al momento in cui termina l'input su un controllo visuale (Control.LostFocus). Questi gestori di dati sono abbastanza flessibili da consentire l'inserimento di un modulo di trasformazione dei dati con cui possiamo, per esempio, trasformare un colore in un indice in una listbox e viceversa, oppure trasformare una data in un formato di visualizzazione diverso da quello che abbiamo nel database.

Il meccanismo del data binding consente

il collegamento con variabili, proprietà di oggetti, interrogazioni XQuery su dataset xml e recordset estratti dal database. Non siamo limitati a operazioni scalari: possiamo anche fare il binding di Collection con un array di controlli, per esempio per trasformare un recordset in una successione di controlli utente. Per fare un esempio, potremmo associare una lista di film in programmazione in una multisala in una serie di controlli utente composti da locandina, titolo, trama e comandi per l'acquisto biglietti.

CONCLUSIONI

Con questa breve introduzione, crediamo di avere dimostrato che ci sono ottime ragioni per considerare l'utilizzo di Xaml, una chiave dello sviluppo up to date con Windows in tutte le versioni, dal cellulare, al tablet, al computer. Consideriamo anche che la recente apertura del codice di .net darà alla piattaforma la possibilità di migrare su altri sistemi. Quello che offre Xaml non è solo, come abbiamo visto, la comodità e l'economia di codice, grazie ai binding, ma anche la razionalizzazione del modello applicativo con una buona separazione fra il codice di interfaccia e il codice legato alle regole di business, adottando uno dei numerosi framework Mvvm (model, view, viewmodel). Infine, una motivazione interessante per passare sul nuovo modello applicativo, lasciando il territorio familiare di WinForms, è la possibilità di entrare in nuovi ambiti, sviluppando contemporaneamente per le diverse versioni di Windows, anche i tablet economici WinRT con processore Arm, e per Windows Phone.

la storia di Windows Presentation Foundation (Wpf) e xaml, illustrata e navigabile, all'indirizzo <http://japf.github.io/xaml-history/>

