

Xamarin di offrire ambienti operativi Windows su Android e iOS, così come ha permesso di creare un ambiente Windows per il piccolo Raspberry PI. Parlando di applicazioni Windows che potranno girare su telefoni con display da 5", tablet da 7" a 10" o computer con schermi da 21" a 27", dobbiamo citare l'ultima novità di spicco. Il codice delle app sarà universale e si adatterà automaticamente. L'architettura basata sul linguaggio di descrizione delle interfacce Xaml, che Microsoft ha adottato da parecchi anni, consente la separazione fra codice e presentazione. La presentazione sarà resa adattiva con l'estensione di Xaml per supportare descrizioni condizionali dell'interfaccia, in modo da permettere di creare interfacce utente che scalano dal video da 27 pollici fino al telefonino, adattandosi alle dimensioni del video.

La conseguenza di questa organizzazione del codice è che le applicazioni Microsoft per il telefono e il computer non saranno versioni diverse dello stesso programma. Un'unica versione, per esempio, di Word potrà sfruttare completamente il display sulla scrivania e lo schermo nella tasca. Di più, l'applicazione in esecuzione scalerà automaticamente da una presentazione all'altra nel momento in cui si connette, per esempio, un telefono a un desktop per utilizzarne video e tastiera. In sostanza, parlando di applicazioni universali Microsoft intende universali in senso stretto.

Cominciamo, allora, a ricapitolare i progetti che vanno sotto il nome di Astoria, Islandwood, Centennial, Westminster, che Microsoft ha annunciato al BUILD 2015, la conferenza annuale per sviluppatori. Le iniziali dei progetti, A, C, I e W, presumibilmente stanno per Android, Com, iOS e Web.

Il progetto Astoria riguarda Android, il sistema operativo mobile più diffuso, che gode di attenzione da Microsoft da diversi mesi. Abbiamo visto, per cominciare, un ottimo emulatore basato sullo strato di virtualizzazione principale di Windows: Hyper-V, che quindi può coesistere con altri emulatori, per

Windows su iPhone? Sì, almeno per noi. Non parliamo, naturalmente, di un sistema operativo alternativo per il telefonino con la mela, ma di una collezione di soluzioni che Microsoft sta aprendo, per fare transitare gli sviluppatori da una piattaforma all'altra.

Chi ha una app per Android o iOS, in sostanza, potrà portarsi dietro il codice, rispettivamente Java o Objective-C, compilarlo in Visual Studio e eseguirlo nativamente, grazie a una libreria di supporto run time nativa Windows, ma con le interfacce dell'ambiente di provenienza.

Per sintetizzare avremo un ambiente esecutivo iOS o Android sui futuri telefonini Windows 10, così come Microsoft offrirà un ambiente run time Windows nativo su Android e iOS (con tutto lo stack .Net), offrendo alle app la possibilità di girare anche su sistemi operativi tradizionalmente considerati competitor dalla stessa Microsoft.

Ai canali che abbiamo descritto, dobbiamo aggiungere un percorso verso Windows Store, per le applicazioni Windows tradizionali, basate su .Net e win32.

Insomma, sia che ci venga in mente di lavorare in Objective-C dentro Visual Studio per portare su Windows

Phone una app scritta per iPhone, sia che si voglia lavorare con C# per costruire una app Android, che usa il framework .Net, potremo contare su un ponte Microsoft su cui transitare. Aprendo queste strade, Microsoft si propone di convogliare applicazioni verso la sua piattaforma, che non è inferiore alle concorrenti per usabilità o efficienza, ma è decisamente indietro come quota di mercato e quindi attira pochi sviluppatori.

Per uno scherzo del destino, quindi, quello che era considerato il più arcano custode della sua piattaforma proprietaria è diventato il motore dello sviluppo *write once, run everywhere*, che è stato dimenticato da tempo nel Java in versione Google e non è mai stato nemmeno un pensiero remoto in casa Apple, meno che mai quando uno sviluppatore iOS sa di lavorare per circa ottocento milioni di device nel mondo (secondo i dati di TheVerge). Si tratta anche del successo dell'idea di distribuire apertamente il proprio codice. Microsoft, infatti, non avrà difficoltà a integrare l'ambiente Android o il compilatore Objective X, entrambi open source.

La distribuzione dell'intero framework .Net su Github, di cui abbiamo parlato in passato, consente a aziende come

esempio quello di Windows Phone, e consente il debugging all'interno di Visual Studio.

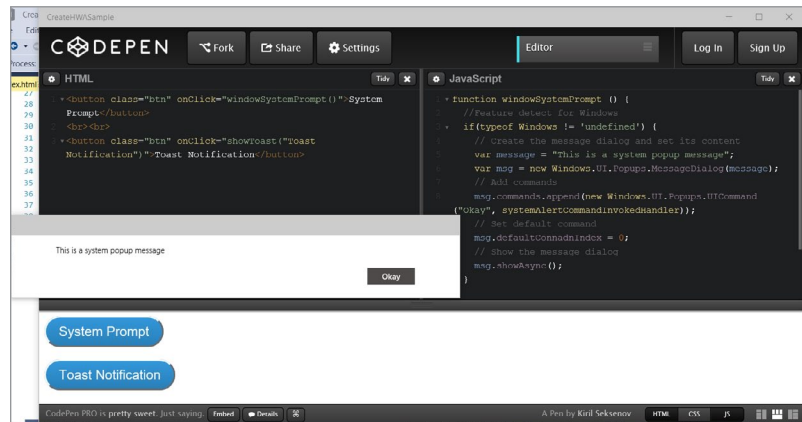
Project Astoria aggiunge uno strato di emulazione delle librerie applicative di Android, che consentirà di portare in Visual Studio il codice Android sviluppato con una delle catene di compilazione native, per esempio con Eclipse, e ricompilare l'applicazione per Windows «con un numero limitato di modifiche al codice».

Non ci sono molti dettagli sul sito Microsoft riguardo al progetto, che è ancora in pieno svolgimento, ma ci si può registrare per ricevere notifiche alla pagina dev.windows.com/uwp-bridges/project-astoria.

In attesa di maggiori dettagli, un'analisi architetturale di massima ci convince della fattibilità di questo progetto. Tutti i sorgenti dell'ambiente operativo Android sono aperti e possono essere consultati senza problemi da Microsoft, in particolare il compilatore Java non dovrebbe porre ostacoli al gruppo di compilatori in Microsoft.

Realizzare un clone della libreria, può essere un lavoro piuttosto faticoso per le dimensioni degli oggetti coinvolti, ma è fattibile, sia perché la macchina virtuale nativa di Windows ha diverse similarità con la macchina virtuale Java specifica di Android, sia perché si tratta di qualcosa che in Microsoft hanno già dimostrato di saper fare, quando l'architetto di .Net, neoassunto, realizzò un clone della macchina virtuale Java appoggiato sull'ambiente Com, che allora era l'architettura di riferimento della software house.

Gli unici ostacoli che possiamo immaginare sono il numero di linee di codice, molto alto, e qualche possibile



Ecco l'applicazione web di prova, convertita in app nativa e in esecuzione nel debugger.

incompatibilità architetturale non risolvibile semplicemente.

PROJECT ISLANDWOOD

Islandwood è il ponte dedicato ai programmatori iOS. Come nel caso di Astoria, ha lo scopo di permettere di importare un progetto XCode in Visual Studio, compilarlo con "modifiche minime" e poterlo compilare e debuggare in Visual Studio.

La home page del progetto non è molto ricca di informazioni, ma si accettano beta tester, anche se in numero limitato. Ci si può prenotare per un ingresso alla pagina dev.windows.com/uwp-bridges/project-islandwood.

Mancano ancora molte informazioni chiave, ma il progetto è piuttosto ambizioso in linea di principio, dato che il collegamento fra il codice e le librerie in Objective-C è differente rispetto al modello di C# e Java, poiché viene risolto in fase di run time, piuttosto che dal compilatore.

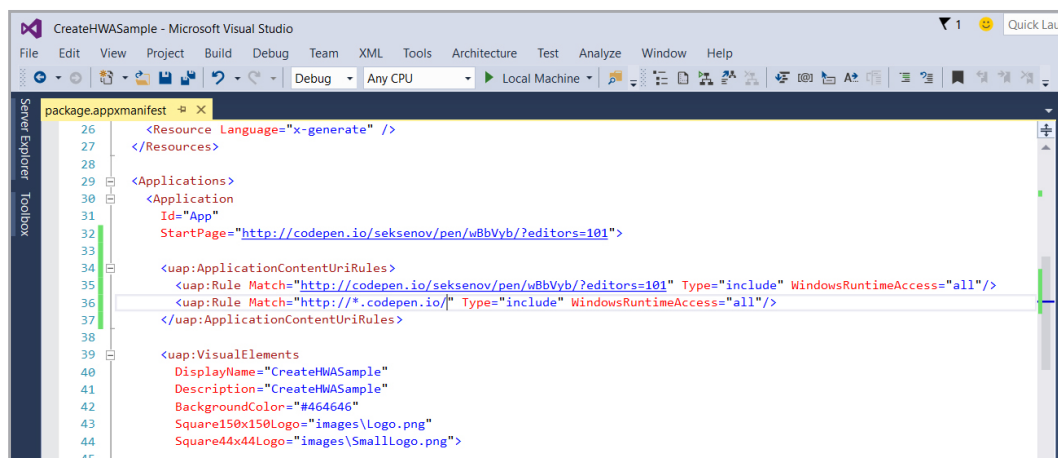
Un altro punto chiave è la sterminata complessità delle librerie Apple, che pone un problema non da poco a chi voglia realizzare un ambiente operativo compatibile.

Alcune risposte si trovano in un video su Channel 9 (channel9.msdn.com/Events/Build/2015/3-610). Una parte interessante è quella dedicata a domande e risposte, che inizia al minuto 58.

Non dubitiamo dell'interesse strategico di Microsoft verso questo bridge, poiché le applicazioni per iOS sono molto numerose e ben curate, ma è difficile sbilanciarsi su cosa sarà effettivamente disponibile questo autunno.

PROJECT WESTMINSTER

Ha la finalità di consentire di distribuire un'applicazione Web nel formato di una app pronta per Microsoft Store. La sua home page si trova all'indirizzo dev.windows.com/uwp-bridges/project-westminster.



Per convertire un'applicazione web in versione pacchettizzata, si aggiungono al manifest dell'applicazione tutte le Url che ne fanno parte.



L'INFORMAZIONE
PIÙ AUTOREVOLE
DAL MONDO
DELL'HI-TECH

SCARICA
LA NOSTRA
APP



Rispetto ai precedenti, questo progetto è in una fase più avanzata, quindi possiamo provare a creare una applicazione seguendo le istruzioni passo per passo alla pagina [microsoftedge.github.io/WebAppsDocs/en-US/win10/CreateHWA.htm](https://github.io/WebAppsDocs/en-US/win10/CreateHWA.htm).

Per iniziare, dobbiamo installare Windows 10 e Visual Studio 2015 RC. Il primo step è solamente la scelta della Url della nostra applicazione; preferibilmente deve essere un sito web che funziona come singola pagina. Se non abbiamo un sito pronto, possiamo usare quello di prova suggerito nella pagina di documentazione.

Si prosegue creando un'applicazione vuota scegliendo, in sequenza le voci JavaScript, Windows, Windows Universal, Blank App (Windows Universal) nell'albero dei template di Visual Studio. Dopo la creazione, cancelliamo il file Default.html e le directory css, js, WinJS dal progetto, perché non ne avremo bisogno. Impostiamo nel manifest dell'applicazione la Url iniziale del programma, sostituendo in modo appropriato la Url nella dichiarazione

```
StartPage="default.html"
```

Aggiungiamo nel manifest le Url che fanno parte dell'applicazione, per esempio come segue

```
<uap:ApplicationContentUriRules>  
  <uap:Rule Type="include"  
    Match="https://example.com/" />  
  <uap:Rule Type="include"  
    Match="https://*.example.com/" />  
  <uap:Rule Type="exclude"  
    Match="https://example.com/  
    excludethispage.aspx" />  
</uap:ApplicationContentUriRules>
```

Questo istruisce l'applicazione su quali sono le pagine interne e quelle che non sono parte del programma, che si apriranno in un browser esterno.

Questo è tutto: siamo già pronti per premere il tasto Run. Naturalmente, quando il codice viene pacchettizzato in questo modo, il codice JavaScript avrà accesso a tutte le funzioni native del device, come l'accelerometro, il Gps, la fotocamera.

PROJECT CENTENNIAL

Questo progetto va a pescare nel bacino più grande di applicazioni su cui Microsoft può contare, cioè le applicazioni per le versioni precedenti di Windows e i precedenti ambienti operativi runtime, Com e Win32. In effetti, si tratta delle applicazioni a cui Microsoft aveva voltato le spalle, quelle che avevano fatto guadagnare a Windows la fama imperitura di sistema operativo non sicuro. La pacchettizzazione delle applicazioni nasce da un approccio nuovo all'installazione. L'ambiente di produzione dell'applicazione monitorizza i cambiamenti al sistema prodotti dal lancio e dall'esecuzione del programma. L'installazione dell'applicazione, quindi viene semplicemente definita come la ricreazione delle differenze nel sistema rispetto a un sistema pulito.

Le applicazioni sono installate nello spazio privato di ogni utente, ed esistono meccanismi per evitare la duplicazione di file. Ogni applicazione mantiene il suo set di differenze rispetto al sistema pulito gira in una sandbox, con un registry separato. Questo dovrebbe superare le difficoltà tipiche delle applicazioni native Win32, come gli effetti secondari di un'applicazione sull'altra. È da prevedere che la fragilità tipica di queste applicazioni si presenti anche nella versione più moderna, ma Windows ha imparato a difendersi dal codice, nel frattempo. Ci dobbiamo aspettare l'ingresso trionfale di moltissime applicazioni, date per obsolete nello Windows Store, grazie a Project Centennial.

Conversion

Il processo di conversione di un'applicazione tradizionale Win32, comincia dalla creazione del pacchetto di installazione, che compila l'elenco di differenze rispetto al sistema base. Queste diventeranno la firma dell'installazione.

