Le Estensioni C++/CX per WinRT

CPP04

Italian C++ Community

Raffaele Rialdi

raffaeler@vevy.com - @raffaeler

http://iamraf.net

Grazie a

Sponsor

icubed

Microsoft

WPC by UnionNet

stackoverflow

aspitalia.com

DomusDotNet

DotDotNet

.net hell .it

.net sland

DotNet Liguria

dotnet LOMBARDIA

dotnet MARCHE

DotNetSide

DOTNET TOSCANA

Italian C++ Community

OrangeDotNet

sharepointcommunity.it

UGIdotNET

Visual Basic tips&tricks

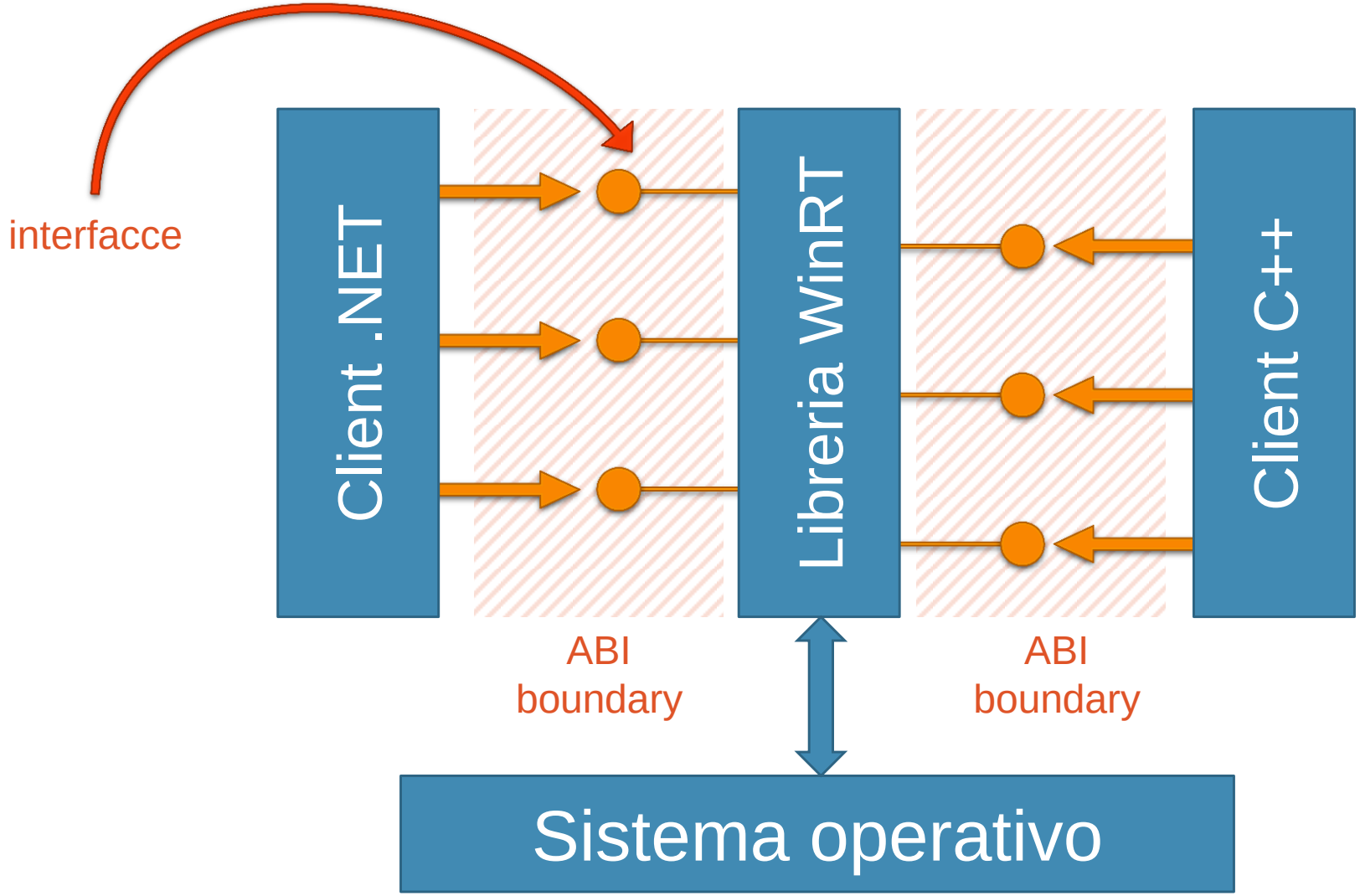
Cuaiss

1nnOva

Perché C++ nelle Store Apps

- Performance
  - Algoritmi consolidati in C++
- Calcolo parallelo
  - Librerie AMP
- Grafica
  - La massima potenza grafica si ottiene solo con DirectX
- Portabilità
  - Mantenere una codebase C++ cross-platform (anche sui device)
  - Si, si può usare ISO C++ su iOS, Android e Windows
  - Portabilità collaudata dagli anni '80 :)

Perché C++/CX



- Non solo Store apps
- Eredità di COM
- Libreria WRL
  Accesso old-style
- C++/CX
  Projection semplice
- Costo della ABI

La "new" e le projection di WinRT

C#

```
var client = new Windows.Web.Syndication.SyndicationClient();
```

C++
WRL

```
HRESULT hr;
ComPtr<IActivationFactory> spFactory;
auto classId = HStringReference(RuntimeClass_Windows_Web_Syndication_SyndicationClient).Get();
hr = GetActivationFactory(classId, &spFactory);
if(FAILED(hr)) return PrintError(hr);
ComPtr<ISyndicationClient> client;
hr = spActivationFactory->ActivateInstance(&client);
if(FAILED(hr)) return PrintError(hr);
```

C++/CX

```
auto client = ref new Windows::Web::Syndication::SyndicationClient();
```

- Le projection sono scorciatoie dei linguaggi che generano codice infrastrutturale
- Gli **smart pointer** (ComPtr) gestiscono il ciclo di vita dell'oggetto WinRT

Type System

Tipi base
Interfacce
String
Type System
Collections
Reference types
Value Types

Partial classes

- Nuovo modificatore "partial"
- Permette di splittare la definizione di classe su più file ".h"
  - In standard C++ è legale usare più .cpp per lo stesso .h
- Indispensabile per il codice generato dai designer

Basic1.xaml.h

```
#include "Basic1.g.h"

namespace CppCXApp
{
    [Windows::Foundation::Metadata::WebHostHidden]
    public ref class Basic1 sealed
    {
    public:
        Basic1();
    private:
        void Go_Click( … );
    };
}
```

Basic1.g.h

```
namespace CppCXApp
{
    partial ref class Basic1 : public ::Windows::UI::Xaml::Controls::Page,
        public ::Windows::UI::Xaml::Markup::IComponentConnector
    {
    public:
        void InitializeComponent();
        virtual void Connect(int connectionId, ::Platform::Object^ target);
    private:
        bool _contentLoaded;
        private: ::Windows::UI::Xaml::Controls::ListBox^ listBox1;
    };
}
```

value class e value struct

- Contiene solo tipi di base e solo value types
  - Non può contenere interfacce WinRT (che identificano i reference type)
  - I costruttori sono forniti solo dal compilatore

```
auto color2 = Windows::UI::Color(color);
▲ 1 of 2 ▼ Windows::UI::Color(const Windows::UI::Color &)
```

```
Windows::UI::Color color;
color.R = 0xff;
color.G = 0;
color.B = 0;
color.A = 0xff;
```

- Solo tipi pubblici WinRT
  - Compresi nullable types Platform::IBox<T>^

- Può contenere (ed esporre) stringhe
  - Platform::String^

ref class e ref struct

- Dichiarabili solo dentro un namespace
  - Il template di VS delle nuove classi non crea il namespace!
- Può implementare una o più interfacce (interface class o interface struct)
- La superficie **pubblica** può contenere **solo** value e ref class di tipo WinRT
- La superficie **privata** può contenere **anche** tipi standard C++
- Se esiste un distruttore pubblico il compilatore implementa Platform::IDisposable (Windows::Foundation::IClosable)
- Non possono essere dichiarate generiche
  - Se private, possono dichiarare template

Hat ^

- È sinonimo di smart pointer
  - Gestisce il reference count

```
SyndicationClient ^ client = ref new SyndicationClient();
```

```
auto client = ref new SyndicationClient();
```

- La creazione dell'istanza è demandata alla **factory** via ref new
- «client» è lo smart pointer
  - Si usa come fosse un vero puntatore (->)

```
auto feedOperation = client->RetrieveFeedAsync(ref new Uri("http://www.iamraf.net/rss"));
```

- La "delete client;" non è mai necessaria
  - Se chiamata invoca il distruttore (l'oggetto diventa invalido)
  - La memoria viene rilasciata solo dopo che il refcount va a zero

ref class e le gerarchie

- Se è una public ref class compare nei metadati
  - può derivare da public ref class private definite nello stesso componente
  - può derivare dalla gerarchia XAML
  - deve essere sealed (a meno che non derivi da DependencyObject)
  - non possono essere template
  - I tipi pubblici:
    - Devono essere dichiarati dentro un namespace
    - Non possono essere classi annidate
    - Il loro namespace deve essere uguale a quello del file winmd in cui sono creati
- Se è una private ref class NON compare nei metadati
  - può non essere sealed (ma non è visibile al di fuori della dll)
  - utile per separare funzionalità
  - possono essere dichiarate come template
- ref class all'interno di una stessa dll non usano la ABI
  - boost di performance importante

Regole sulle gerarchie

- Single inheritance sulle classi
- Multiple Inheritance sulle interfacce
- Se il costruttore è pubblico, la ref class deve essere sealed
  - A meno che non derivi da DependencyObject
- La ref class può essere pubblica + non-sealed
  - Se deriva da DependencyObject
  - Se ha un costruttore internal / protected private
- La private inheritance è proibita
  - Previene problemi da classi base di tipo COM

Visibilità nelle ref class/struct

	Modificatore	Visibilità dentro il componente	Presente in winmd	Visibilità per chi lo referenzia
Standard C++	private	No	No	No
	protected	protected	Si	protected
	public	public	Si	public
Microsoft C++/CX	internal	public	No	No
	protected private (private protected)	protected	No	No
	public protected (protected public)	public	Si	protected

Properties

- Triviali
  - Backing store automatico

```
property Platform::String^ Name1;
```

- Esplicite
  - visibilità differenziabile

```
private:
   Platform::String^ _name2;
public:
   property Platform::String^ Name2
   {
      Platform::String^ get() { return _name2; }
//private:
      void set(Platform::String^ value) { _name2 = value; }
   }
```

- Read-only

```
private:
   Platform::String^ _name2;
public:
   property Platform::String^ Name2
   {
      Platform::String^ get() { return _name2; }
   }
```

```
Platform::String^ wstring2RT(std::wstring wstr)
{
    auto rts = ref new Platform::String(wstr.c_str(), wstr.size());
    return rts;
}
```

Platform::String

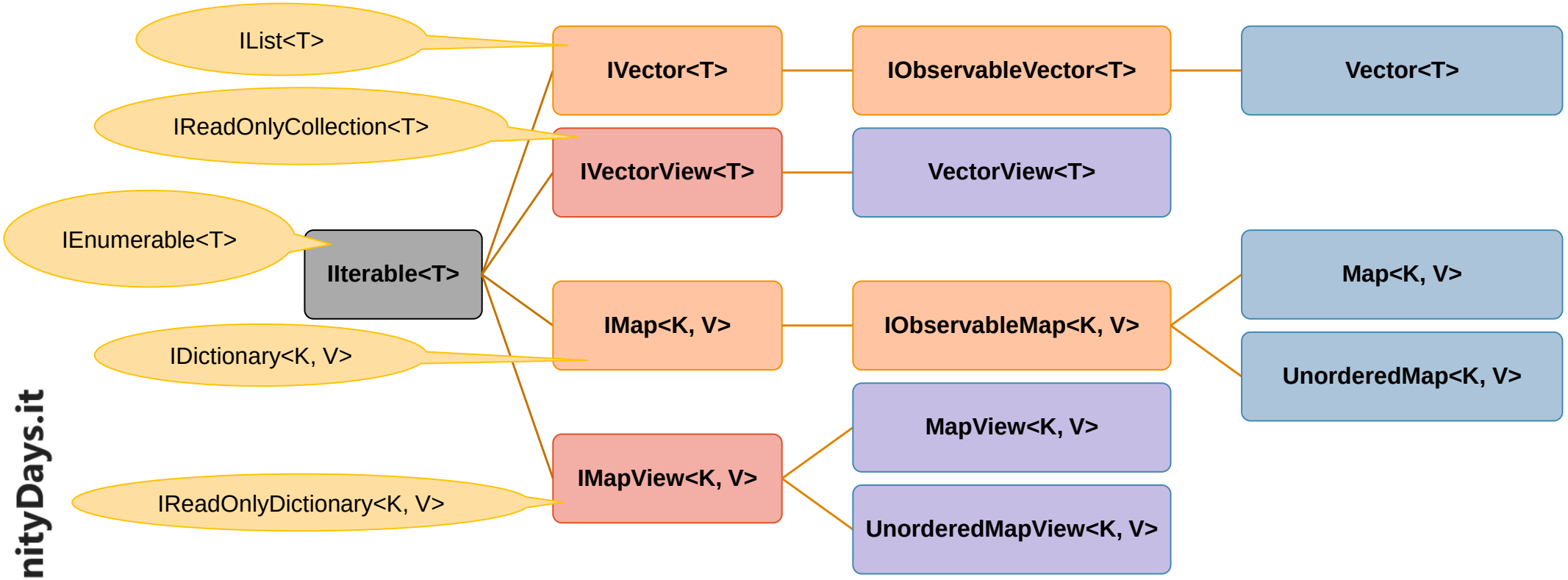
- Value-Type, non accetta null
  - È "speciale", viene istanziato con ref new e referenziato con ^
  - Internamente gestisce la modalità "Fast Pass" che sospende il refcount
- Sequenza di UTF-16 (char16 == wchar_t) immutabile, può contenere null
  - Stesso formato binario di .NET
  - L'immutabilità è amica del codice multitherading
- Per costruire stringhe usare wstring oppure vector<wchar_t>
  - wstring ➔ insert, push_back, pop_back, replace, +=
  - vector ➔ push_back, pop_back, at, [ ], front, back
- Best practice:
  - Usare Platform::String^ solo sulla ABI
  - Per usare begin(str), end(str) includere <vccorlib.h>

Platform::StringReference

```
std::wstring w(L"Hello, world");
Platform::StringReference r1(w.c_str());
```

- È un wrapper su un buffer di wchar_t
  - il buffer NON viene copiato
- È un tipo C++ nativo, non un componente WinRT
  - Non può essere pubblicato nei metadati
- Può essere usato al posto di Platform::String^ laddove evitare la copia dei dati comporta un guadagno in performance
- Attenzione:
  - il buffer originale deve rimanere valido fino a che viene usato
  - il buffer originale non deve mutare

Le collections e le rispettive interfacce



- #include <collections.h> è fondamentale per beneficiare dell'integrazione STL

Collections tips

- Sulla ABI usare solo le interfacce
  - No ➔ Vector<String^>^
  - Si ➔ IObservableVector<String^>^
- Internamente usare std::vector<T> (maggiore efficienza)
  - Usare std::move per trasferire (non copiare) il contenuto in un Vector<T>
- Vector<T> pretende che T sia comparabile
  - L'implementazione custom può essere fatta con std::equal_to
- Sulla ABI la collection viene passata come handle / reference
  - Il costo di richiesta di ciascun Item è alto (attraversamento della ABI)
  - Può convenire copiare tutti gli elementi in un DTO di .NET

Platform::Array

- Array è l'analogo di std::array
  - ArrayReference è un facet su Array (analogo di StringReference)
- Per passare un array di dati sulla ABI:
  - Prediligere la gerarchia IVector<T>
  - "PassArray" const Platform::Array^
    - Quando il chiamante passa un array al chiamato
  - "FillArray" Platform::WriteOnlyArray<T>
    - Quando il chiamante passa un array che deve essere riempito
  - "ReceiveArray" Platform::Array^
    - Quando il chiamato ritorna un array al chiamante

Async Pattern

Async e PPL

```
task<void> CppCXApp::Basic1::FillList(String^ url)
{
  auto client = ref new SyndicationClient();
  auto uri = ref new Uri(url);

  return create_task(client->RetrieveFeedAsync(uri))
    .then( [this] (SyndicationFeed^ feed)
    {
      for (auto item : feed->Items)
      {
        this->listBox1->Items->Append(item->Title->Text);
      }
    } );
}
```

Q&A

Tutto il materiale di questa sessione su

http://www.communitydays.it/

Lascia il feedback su questa sessione,

potrai essere estratto per i nostri premi!

Seguici su

Twitter @CommunityDaysIT

Facebook http://facebook.com/cdaysit

#CDays14