

Recursos Avançados de C++

Módulo 3

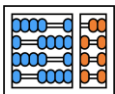
Prof. Dr. Bruno B. P. Cafeo

Instituto de Computação
Universidade Estadual de Campinas

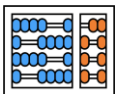


Agenda

- Conceitos básicos do COM
- ATL
- MFC

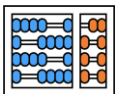


Conceitos básicos do COM



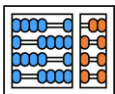
Introdução ao COM

- O Component Object Model (COM) é um Modelo de Objeto de Componente desenvolvido pela Microsoft.
- É um padrão de interoperabilidade binária que permite a reutilização de bibliotecas de software em diferentes aplicativos, independentemente da linguagem de programação utilizada.



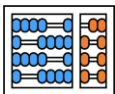
Princípios Fundamentais

- Padrão Binário: O COM estabelece um padrão binário para chamadas de função entre componentes.
- Interfaces: Componentes COM agrupam funções em interfaces, permitindo polimorfismo e descoberta de recursos.
- Identificação Única: Cada interface é identificada por um GUID (IID), garantindo exclusividade.
- Carregador de Componentes: O COM gerencia o carregamento de componentes em tempo de execução.



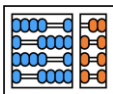
Interface IUnknown

- Base de Todas as Interfaces: IUnknown é a base de todas as interfaces COM.
- Três Funções Membro: IUnknown define três funções membros: QueryInterface, AddRef e Release.
- Contagem de Referência: O controle de tempo de vida dos objetos COM é feito por meio da contagem de referência.
- Uso de QueryInterface: A função QueryInterface permite a navegação entre interfaces.




Interface IUnknown

```
1 // Definindo a interface ICalculator
2 struct ICalculator : public IUnknown
3 {
4     virtual HRESULT __stdcall Add(int a, int b, int* result) = 0;
5     virtual HRESULT __stdcall Subtract(int a, int b, int* result) = 0;
6 };
7
```



Interface IUnknown

No PowerShell -> [System.Guid]::NewGuid()



```
1 // Exemplo de IDL para a interface ICalculator
2 import "oaidl.idl";
3 import "ocidl.idl";
4
5 [
6     object,
7     uuid(XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX),
8     dual,
9     helpstring("ICalculator Interface"),
10    pointer_default(unique)
11 ]
12 interface ICalculator : IDispatch
13 {
14     [id(1), helpstring("method Add")] HRESULT Add([in] int a, [in]
15         int b, [out, retval] int* result);
16     [id(2), helpstring("method Subtract")] HRESULT Subtract([in]
17         int a, [in] int b, [out, retval] int* result);
18 }
```


Interface IUnknown

```
1 // Definindo a classe Calculator
2 class Calculator : public ICalculator
3 {
4 private:
5     long m_refCount;
6
7 public:
8     Calculator() : m_refCount(1) {}
9
10    // Implementação do método Add
11    HRESULT __stdcall Add(int a, int b, int* result) override
12    {
13        *result = a + b;
14        return S_OK;
15    }
16
17    // Implementação do método Subtract
18    HRESULT __stdcall Subtract(int a, int b, int* result) override
19    {
20        *result = a - b;
21        return S_OK;
22    }
23 }
```

```
24 // Implementação da IUnknown
25 ULONG __stdcall AddRef() override
26 {
27     return InterlockedIncrement(&m_refCount);
28 }
29
30 ULONG __stdcall Release() override
31 {
32     if (InterlockedDecrement(&m_refCount) == 0)
33     {
34         delete this;
35         return 0;
36     }
37     return m_refCount;
38 }
39
40 HRESULT __stdcall QueryInterface(REFIID riid, void** ppv)
41     override
42 {
43     if (riid == IID_IUnknown || riid == IID_ICalculator)
44     {
45         *ppv = this;
46         AddRef();
47         return S_OK;
48     }
49     *ppv = nullptr;
50     return E_NOINTERFACE;
51 }
```

* Incluir nesse arquivo o arquivo gerado a partir da compilação do IDL

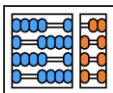
INF1900 – Prof. Bruno Cafeo | cafeo@ic.unicamp.br | 2023s2

Registro do servidor

```
1 // Funções para registrar e desregistrar o servidor
2 static HRESULT RegisterServer()
3 {
4     // Registre as informações da classe no Registro do Windows
5     return _Module.RegisterServer(TRUE);
6 }
7
8 static HRESULT UnregisterServer()
9 {
10    // Remova as informações da classe do Registro do Windows
11    return _Module.UnregisterServer(TRUE);
12 }
13
```

Registrar -> regsvr32 NomeDaSuaDLL.dll

“Desregistrar” -> regsvr32 /u NomeDaSuaDLL.dll



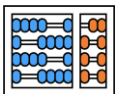
Cliente

```
1 int main() {  
2     // Inicializa o ambiente COM  
3     CoInitialize(nullptr);  
4  
5     // Cria uma instância do objeto COM a partir do servidor  
6     ICalculator* calculator = nullptr;  
7     HRESULT hr = CoCreateInstance(CLSID_Calculator, nullptr,  
8                                     CLSCTX_INPROC_SERVER, IID_ICalculator, (void**)&calculator);  
9  
10    if (SUCCEEDED(hr)) {  
11        // Chama os métodos do objeto COM  
12        int result;  
13        calculator->Add(5, 3, &result);  
14        std::cout << "Resultado da adição: " << result << std::endl;  
15  
16        calculator->Subtract(10, 7, &result);  
17        std::cout << "Resultado da subtração: " << result << std::endl;  
18  
19        // Libera o objeto COM quando não for mais necessário  
20        calculator->Release();  
21    } else {  
22        std::cerr << "Erro ao criar uma instância do objeto COM" << std::endl;  
23    }  
24  
25    // Finaliza o ambiente COM  
26    CoUninitialize();  
27  
28    return 0;  
29 }
```

* Incluir nesse arquivo o .h gerado da compilação da DLL

Modelo Cliente/Servidor

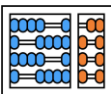
- Implementação de Múltiplas Interfaces: Um objeto COM pode implementar várias interfaces.
- Banco de Dados de Registro: O COM mantém um banco de dados de registro de CLSIDs para os objetos instalados.
- Relação Cliente/Servidor: A interação entre objetos COM e chamadores segue o modelo cliente/servidor.
- Função CoCreateInstance: A função CoCreateInstance cria instâncias de objetos COM.



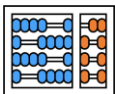
Cliente utilizando Servidor (IClassFactory)

```
1 #include <windows.h>
2 #include "ICalculator.h" // Inclua o cabeçalho gerado
3
4 int main() {
5     // Inicialização do ambiente COM
6     CoInitialize(nullptr);
7
8     // Obtenha uma instância da classe factory do servidor COM
9     IClassFactory* classFactory = nullptr;
10    HRESULT hr = CoGetClassObject(CLSID_Calculator,
11    CLSCTX_LOCAL_SERVER, NULL, IID_IClassFactory, (void*)&
12    classFactory);
13
14    if (SUCCEEDED(hr)) {
15        // Crie uma instância da classe Calculator usando a classe
16        // factory
17        ICalculator* calculator = nullptr;
18        hr = classFactory->CreateInstance(nullptr, IID_ICalculator, (
19        void*)&calculator);
20
21        if (SUCCEEDED(hr)) {
22            // Chame os métodos da interface ICalculator
23            int result;
24            hr = calculator->Add(5, 3, &result);
25
26            if (SUCCEEDED(hr)) {
27                // Exiba o resultado da adição
28                printf("Resultado da adição: %d\n", result);
29            } else {
30                printf("Erro ao chamar o método Add\n");
31            }
32        }
33    }
34}
```

```
29 hr = calculator->Subtract(10, 7, &result);
30
31 if (SUCCEEDED(hr)) {
32     // Exiba o resultado da subtração
33     printf("Resultado da subtração: %d\n", result);
34 } else {
35     printf("Erro ao chamar o método Subtract\n");
36 }
37
38 } else {
39     printf("Erro ao criar uma instância da classe
40     Calculator\n");
41 }
42
43 // Libere a classe factory
44 classFactory->Release();
45 } else {
46     printf("Erro ao obter a classe factory do servidor COM\n");
47 }
48
49 // Finalização do ambiente COM
50 CoUninitialize();
51
52 return 0;
53 }
```

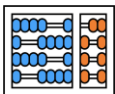


ATL



Introdução ao ATL

- A ATL é uma biblioteca de modelos C++ que fornece um conjunto de classes e macros para simplificar o desenvolvimento de software para Windows.
- Foi criada para atender às necessidades específicas de desenvolvedores que trabalham com Component Object Model (COM) e ActiveX.
- É altamente otimizada para desempenho e eficiência.

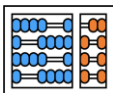


Calculator.idl

No PowerShell -> [System.Guid]::NewGuid()

```
1 // Exemplo de IDL para a interface ICalculator
2 import "oaidl.idl";
3 import "ocidl.idl";
4
5 [
6     object,
7     uuid(XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX),
8     dual,
9     helpstring("ICalculator Interface"),
10    pointer_default(unique)
11 ]
12 interface ICalculator : IDispatch
13 {
14     [id(1), helpstring("method Add")] HRESULT Add([in] int a, [in]
15     int b, [out, retval] int* result);
16     [id(2), helpstring("method Subtract")] HRESULT Subtract([in]
17     int a, [in] int b, [out, retval] int* result);
18 }
```

- **import "oaidl.idl" e import "ocidl.idl"**: Importa definições de tipos e interfaces que são usadas comumente em componentes COM, incluindo tipos OLE Automation e tipos de controle OLE.
- **object**: Define que a interface é um objeto.
- **uuid(12345678-1234-1234-1234-123456789012)**: Atribui um identificador UUID exclusivo à interface. O UUID é usado para identificar exclusivamente a interface.
- **dual**: Indica que a interface suporta chamadas de método de dispatch (IDispatch).
- **pointer_default(unique)**: Define que o ponteiro para o objeto da interface é único.

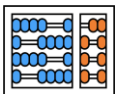


Calculator.h

Gerado automaticamente pelo Visual C++

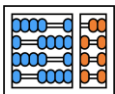
```
1  #pragma once
2  #include "resource.h"
3  #include "Calculator_i.h"
4
5  class ATL_NO_VTABLE CCalculator :
6  public CComObjectRootEx<CComSingleThreadModel>, // Classe base para gerenciamento
7  de vida e interfaces
8  public CComCoClass<CCalculator, &CLSID_Calculator>, // Informações de classe COM
9  public IDispatchImpl<ICalculator, &IID_ICalculator, &LIBID_CalculatorLib, /*wMajor
10  =*/ 1, /*wMinor =*/ 0> // Implementação de IDispatch
11 {
12 public:
13     CCalculator() {}
14
15     DECLARE_REGISTRY_RESOURCEID(IDR_CALCULATOR) // Registro de informações
16
17     DECLARE_NOT_AGGREGATABLE(CCalculator) // Não suporta agregação
18
19     BEGIN_COM_MAP(CCalculator)
20     COM_INTERFACE_ENTRY(ICalculator) // Mapeamento da interface ICalculator
21     COM_INTERFACE_ENTRY(IDispatch) // Mapeamento da interface IDispatch
22     END_COM_MAP()
23
24     // ICalculator
25 public:
26     STDMETHOD(Add)(LONG a, LONG b, LONG* result);
27     STDMETHOD(Subtract)(LONG a, LONG b, LONG* result);
28 };
```

- **CComObjectRootEx**: Classe base para gerenciamento de vida e interfaces. É usada para gerenciar a contagem de referências e implementações de interfaces.
- **CComCoClass**: Define informações de classe COM, incluindo o CLSID (identificador de classe).
- **IDispatchImpl**: Implementação da interface IDispatch, que é usada para suportar automação.
- **DECLARE_REGISTRY_RESOURCEID**: Define informações de registro para o servidor COM, como CLSID e descrições.
- **DECLARE_NOT_AGGREGATABLE**: Indica que o objeto não suporta agregação.
- **BEGIN_COM_MAP**: Mapeia as interfaces suportadas pelo objeto.
- **COM_INTERFACE_ENTRY**: Define as interfaces que o objeto suporta.



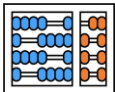
Calculator.cpp

```
1  #include "stdafx.h"
2  #include "Calculator.h"
3
4  STDMETHODIMP CCalculator::Add(LONG a, LONG b, LONG* result)
5  {
6      if (result == nullptr)
7          return E_POINTER;
8
9      *result = a + b;
10     return S_OK;
11 }
12
13 STDMETHODIMP CCalculator::Subtract(LONG a, LONG b, LONG* result)
14 {
15     if (result == nullptr)
16         return E_POINTER;
17
18     *result = a - b;
19     return S_OK;
20 }
21
```

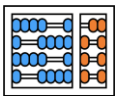


Client.cpp

```
1  #include <atlbase.h>
2  #include "Calculator_i.h"
3
4  int main() {
5      // Inicialização do ambiente COM
6      CoInitialize(nullptr);
7
8      {
9          CComPtr<IClassFactory> classFactory;
10         HRESULT hr = CoGetClassObject(CLSID_CalculatorFactory, CLSCTX_LOCAL_SERVER,
11             NULL, IID_IClassFactory, (void**)&classFactory);
12
13         if (SUCCEEDED(hr)) {
14             CComPtr<ICalculator> calculator;
15             hr = classFactory->CreateInstance(nullptr, IID_ICalculator, (void**)&
16                 calculator);
17
18             if (SUCCEEDED(hr)) {
19                 long result;
20                 hr = calculator->Add(5, 3, &result);
21
22                 if (SUCCEEDED(hr)) {
23                     printf("Resultado da operação de adição: %ld\n", result);
24                 } else {
25                     printf("Erro ao chamar o método Add\n");
26                 }
27             } else {
28                 printf("Erro ao criar uma instância da classe Calculator\n");
29             }
30         } else {
31             printf("Erro ao obter a classe factory do servidor COM\n");
32         }
33     }
34
35     // Finalização do ambiente COM
36     CoUninitialize();
37
38     return 0;
39 }
```

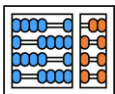


MFC



Introdução ao MFC

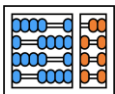
- O MFC (Microsoft Foundation Classes) é uma biblioteca C++ fornecida pela Microsoft para desenvolvimento de aplicativos Windows.
- Facilita a criação de aplicativos Windows com uma interface gráfica de usuário (GUI) baseada em janelas.



Cabeçalho

```
1 #include "afxwin.h"  
2
```

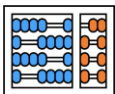
- Cabeçalho principal para desenvolvimento de aplicativos MFC (Microsoft Foundation Classes) no Visual C++.
- Este cabeçalho contém definições para classes e funções essenciais do MFC.



Classe CMyApp

```
1 class CMyApp : public CWinApp
2 {
3     public:
4         virtual BOOL InitInstance();
5     };
6
```

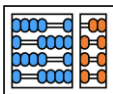
- Definindo uma classe `CMyApp` que é derivada da classe `CWinApp`
- `CWinApp` é uma classe base para aplicativos MFC e fornece funcionalidades de inicialização.
- A função `InitInstance` será substituída para personalizar a inicialização do aplicativo.



Classe CMyWnd

```
1 class CMyWnd : public CFrameWnd
2 {
3 public:
4     CMyWnd()
5     {
6         Create(NULL, _T("Exemplo MFC com Botão"), WS_OVERLAPPEDWINDOW, CRect(100,
7             100, 400, 300));
8         button.Create(_T("Clique-me"), WS_CHILD | WS_VISIBLE, CRect(10, 10, 100,
9             30), this, 1);
10    }
11
12    afx_msg void OnButtonClick();
13    DECLARE_MESSAGE_MAP()
14 private:
15     CButton button;
16 };
```

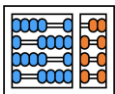
- CMyWnd representa a janela principal do aplicativo.
- CFrameWnd é usada para criar janelas com bordas.
- Create(...): Cria a janela principal do aplicativo com um título, estilo de janela e posição na tela.
- button.Create(...): Cria um botão na janela, especificando seu texto, estilo e posição. O botão é associado à janela atual (this) com um identificador de controle (1).
- afx_msg void OnButtonClick(): Esta é uma declaração de função para manipular o evento de clique do botão. A macro afx_msg é usada para indicar que esta função é um manipulador de mensagens do MFC.
- DECLARE_MESSAGE_MAP(): Declara um mapa de mensagens para a classe CMyWnd. O mapa de mensagens é usado para associar eventos (como cliques de botão) a funções de manipulação.



Implementando a criação de uma instância de CMyApp

```
1  BOOL CMyApp::InitInstance()  
2  {  
3      m_pMainWnd = new CMyWnd();  
4      m_pMainWnd->ShowWindow(SW_SHOW);  
5      m_pMainWnd->UpdateWindow();  
6      return TRUE;  
7  }  
8
```

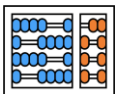
- Criando uma instância da classe `CMyWnd`, que representa a janela principal do aplicativo.
- Estamos exibindo a janela com `ShowWindow` e atualizando-a com `UpdateWindow`.
- Por fim, retornamos `TRUE` para indicar que a inicialização foi bem-sucedida



Mapa de mensagens

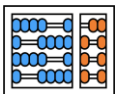
```
1 BEGIN_MESSAGE_MAP (CMyWnd, CFrameWnd)
2     ON_COMMAND(1, OnButtonClick)
3 END_MESSAGE_MAP ()
4
```

- Mapa de mensagens para a classe CMyWnd.
- Associando o evento de clique do botão (identificado pelo valor 1) à função OnButtonClick.



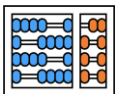
Implementação do clique do botão

```
1 void CMyWnd::OnButtonClicked()  
2 {  
3     AfxMessageBox(_T("Botão foi pressionado!"));  
4 }  
5
```



Declarando a aplicação

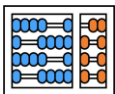
```
1 CMyApp theApp;  
2
```



Código completo

```
1  #include "afxwin.h"
2
3  class CMyApp : public CWinApp
4  {
5  public:
6      virtual BOOL InitInstance();
7  };
8
9  class CMyWnd : public CFrameWnd
10 {
11 public:
12     CMyWnd()
13     {
14         Create(NULL, _T("Exemplo MFC com Botão"), WS_OVERLAPPEDWINDOW,
15             CRect(100, 100, 400, 300));
16         button.Create(_T("Clique-me"), WS_CHILD | WS_VISIBLE, CRect(10, 10,
17             100, 30), this, 1);
18     }
19
20     afx_msg void OnButtonClick();
21     DECLARE_MESSAGE_MAP()
22 private:
23     CButton button;
24 }
```

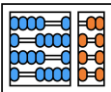
```
25  BOOL CMyApp::InitInstance()
26  {
27      m_pMainWnd = new CMyWnd;
28      m_pMainWnd->ShowWindow(SW_SHOW);
29      m_pMainWnd->UpdateWindow();
30      return TRUE;
31  }
32
33  BEGIN_MESSAGE_MAP(CMyWnd, CFrameWnd)
34      ON_COMMAND(1, OnButtonClick)
35  END_MESSAGE_MAP()
36
37  void CMyWnd::OnButtonClick()
38  {
39      AfxMessageBox(_T("Botão foi pressionado!"));
40  }
41
42  CMyApp theApp;
43
```

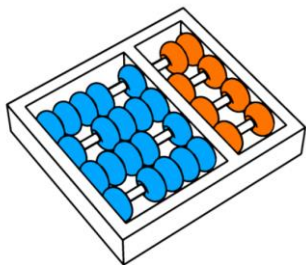


E em Win32 API?

```
1 #include <windows.h>
2
3 // Declaração de função de procedimento da janela
4 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam
5 );
6
7 int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
8 lpCmdLine, int nCmdShow)
9 {
10     // Registrar a classe da janela
11     WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_HREDRAW | CS_VREDRAW, WndProc,
12         0L, 0L, GetModuleHandle(NULL), NULL, NULL, NULL, NULL, _T(
13         "MyWindowClass"), NULL };
14     RegisterClassEx(&wc);
15
16     // Criar a janela
17     HWND hwnd = CreateWindow(_T("MyWindowClass"), _T("Exemplo sem MFC"),
18         WS_OVERLAPPEDWINDOW, 100, 100, 400, 300, NULL, NULL, hInstance, NULL);
19
20     // Criar um botão na janela
21     HWND button = CreateWindow(_T("BUTTON"), _T("Clique-me"), WS_CHILD |
22         WS_VISIBLE, 10, 10, 100, 30, hwnd, NULL, hInstance, NULL);
23
24     // Mostrar a janela
25     ShowWindow(hwnd, nCmdShow);
26     UpdateWindow(hwnd);
27
28     // Loop de mensagens
29     MSG msg;
30     while (GetMessage(&msg, NULL, 0, 0))
31     {
32         TranslateMessage(&msg);
33         DispatchMessage(&msg);
34     }
35
36     return msg.wParam;
37 }
```

```
33 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
34 {
35     switch (msg)
36     {
37     case WM_COMMAND:
38         if (LOWORD(wParam) == 1)
39         {
40             MessageBox(hwnd, _T("Botão foi pressionado!"), _T("Mensagem"),
41                 MB_OK | MB_ICONINFORMATION);
42         }
43         break;
44     case WM_DESTROY:
45         PostQuitMessage(0);
46         break;
47     default:
48         return DefWindowProc(hwnd, msg, wParam, lParam);
49     }
50     return 0;
51 }
```





**INSTITUTO DE
COMPUTAÇÃO**



Prof. Dr. Bruno B. P. Cafeo

Sala 04
Instituto de Computação - Unicamp
Av. Albert Einstein, 1251
Cidade Universitária
Campinas – SP
13083-852

<https://ic.unicamp.br/~cafeo/>
cafeo@ic.unicamp.br