CMSC433, Fall 2001
COM - Component Object Model

Alan Sussman

December 4, 2001

Administrivia

- Project 6 due Monday, 12/10, and commentary for Project 5
  - grading guidelines posted
- No office hours tomorrow
  - Thursday 2-4
- Classes/Interfaces to look at for Project 6
  - java.awt.Toolkit
  - javax.swing.{JFrame, JMenu, JPanel}
  - java.awt.{Component, Graphics} – repaint()
  - java.awt.event.{ActionListener, MouseListener, MouseAdapter}

Component Object Model

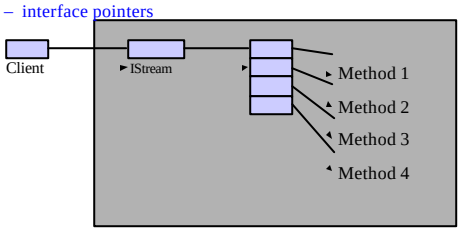
- Language independent
- OS independent (in theory)
- Way to allow components to be designed, deployed, upgraded
  - Need to interact with code written after you were deployed

Immutable interfaces

- Interact through interfaces
  - No direct access to fields
  - Interfaces must never be changed
  - Interfaces assigned a GUID
    - avoid name clashes
    - allow versioning by assigning a new GUID

COM

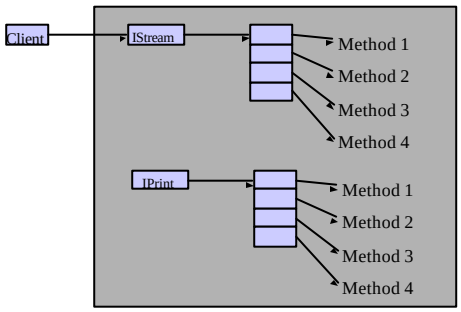
- A binary compatibility standard
  - interface pointers



Multiple interfaces

- Components can implement multiple interfaces
- Different interfaces may correspond to different entry points to object
  - C++ multiple inheritance
  - adaptors

Multiple interfaces



Interfaces in COM

- Similar to interfaces in Java
  - no variables
- Interfaces have a 128 bit Unique ID
  - immutable, never changed, no collisions
- In writing COM code, always use pointers/references to interfaces

Reference counting

- COM objects are reference counted
  - each object keeps track of the number of pointers to it
- When ref count goes to zero, element deletes itself
- Cycles can be a problem
- Remembering where to put all increments and decrements can be a problem

Each interface counted separately

- Each entry point/interface to a COM object is ref counted separately
  - allows an adaptor to be garbage collected

IUnknown

- All COM interfaces must extend interface IUnknown , with methods:
  - HRESULT QueryInterface(const IID& iid, void ** ppv)
  - ULONG AddRef() // inc ref count
  - ULONG Release() // dec ref count

QueryInterface

- Like a C++ dynamic cast
  - Does the component support this interface?
  - If so, gives back a pointer of that kind
    - incrementing the ref count for that interface
  - Else, signal failure

QueryInterface rules

- You always get the same IUnknown
  - multiple queries return the same interface
- You can get an interface if you got it before
  - interfaces don't go away
- You can get the interface you have
  - a component can query itself

HRESULT

- COM doesn't understand exceptions
- So almost all methods return an HRESULT
  - numerical indication of success or a specific error

Creating objects in COM

- Each component has a CLSID (class ID)
- Can call CoCreateInstance
- Can get Class Factory, then create instances directly
- Each DLL has a function that can return class factories for all classes that can be created by that DLL

Smart Pointers

- Automatically take care of reference counting
  - Some versions of COM smart pointers automatically perform dynamic casting (via calls to QueryInterface)
  - Not recommended

Smart Pointers

```
template <class T> class Iptr
    {
    T* p;
    Iptr() : p(0) {};
    Iptr(T* q) : p(q) {
          if (p) p->AddRef();
          }
    Iptr(Iptr<T> q) : p(q.p) {
          if (p) p->AddRef();
          }
    ~Iptr() {
          if (p) p->Release(); }
```

```
    T* operator T*() { return p; }
    T& operator*() { return *p;}
    T* operator->() { return p; }
    T** operator&() {
        assert(p == NULL);
        return &p; }
    //
    // operator = left as exercise
```


FooBar

- interface IFoo : public IUnknown { … }
- interface IBar : public IUnknown { … }
- class Foo : public IFoo { … }
- class Bar : public IBar { … }
- class FooBar : public Foo, public Bar { … }

Implementing QueryInterface

```
STDMETHODIMP QueryInterface(const IID& iid,
                            void **ppv) {
    if (iid == IID_IUnknown || iid == IID_IFoo)
        *ppv = static_cast< Foo*>(this);
    else if (iid == IID_IBar)
        *ppv = static_cast<Bar *>(this);
    else {
        *ppv = null; return E_NOINTERFACE;
        }
    reinterpret_cast<IUnknown *>(*ppv)->AddRef();
    return S_OK;
    }
```


Things to note

- If you support 100 interfaces, cascaded if statements are going to get expensive
  - can't use case statements (UIID's aren't ints)
  - could use custom hashtable
- Separate ref counts
  - could put call to AddRef in each branch
  - would eliminate reinterpret_cast
  - but would increase code size

I want everything

- Why can't I ask
  - what is the list of all of the interfaces you support?
- What would you do with the list of all interfaces a component supports?
- Can use component categories

Component categories

- Assigned a GUID
- Corresponds to a set of interfaces
  - If a component is registered as member of a category
  - instances of that component support all of those interfaces
  - will still need to use QueryInterface to move between interfaces

Categories in Java

- Just define a *Mega-interface*
  - An interface that extends all of the interfaces in the category
  - Ask if class/component implements that
  - Can use reference to Mega-interface type to invoke all methods from any interface in category
  - No casting needed

Component reuse

- How to reuse components?
  - Base class (implementation inheritance)
  - Containment (have as a member)
    - Delegation - Some methods get directly forwarded
    - Adaptor - Some methods get translated
  - Aggregation

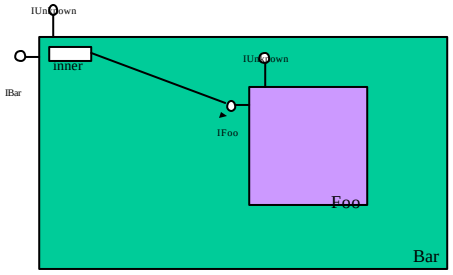
Aggregation

- Say I have a component Bar
  - which uses a component Foo
- Foo implements the IFoo interface
- Bar also implements the IFoo interface
  - by handing things off to its Foo
- Could handle by delegation
  - but that adds an additional level of indirection

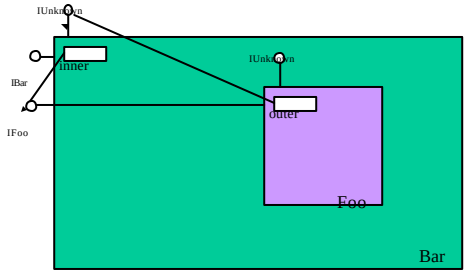
Using Aggregation

- When someone asks a Bar for its IFoo interface
  - just hand them a reference to your Foo
  - handles all IFoo function calls
- But what if you invoke QueryInterface on the IFoo reference and ask for an IBar interface?

Delegation/Forwarding



Aggregation



Supporting aggregation

- You must be able to be told that you have an outer component
- Calls to QueryInterface should be routed to your outer component
- Reference counts are a little tricky
  - cycle could prevent stuff from being collected