Dynamic Invocation in COM
- Can Dynamic invocation in COM be done as in Corba?
  - Use Request object instead of stub
  - marshalling/protocol remains the same
- NO! In case of inproc object there is no stub/skeleton involved
  i.e. call is direct
- Dynamic invocation through the IDispatch interface
- Server must implement the IDispatch interface
- Is used for dynamic programming languages (Visual Basic, scripting languages, ...) 
  - Wizards can help you implement IDispatch
  - IDispatch interface often uses special parameter types: VARIANT, BSTRING, SAFEARRAY
  - These are just complicated struct/unions to represent Visual Basic datatypes.

Client side
- The client can then use this IDispatch interface
  - So the client must be able to call at least one static interface
  - The client must do QueryInterface to get the IDispatch interface
  - method GetIDsOfNames is used to couple names of methods and DISPIDs
  - It has an array of names as in parameter and an array of DISPIDs as out parameter
  - The Invoke method is used to make the call
  - The client calls the Invoke method
  - A special VARIANT type is used for the parameters
  - VARIANT contains type info for the marshaller
  - Visual Basic types can be used and some special structs (BSTRING, SAFEARRAY, DATE, DECIMAL)

Interfaces
- Multiple Interfaces
  - If multiple dispinterfaces are defined by the object, one has to be designated as default.
  - This is what Visual Basic uses.
  - The QueryInterface for IDispatch will give the default interface
  - The others can be obtained by explicit QueryInterface
- Dual Interface
  - A dual interface is an interface that can be called both statically and dynamically
  - It inherits from IDispatch and adds also the static methods
  - This may be important for high performance applications
Contents

- Dynamic Invocation
- Containment/Aggregation/Delegation
- Automation
- Monikers

Implementation inheritance in COM

- Microsoft literature about COM discourages inheritance
- Fragile Base Class problem:
  - Mostly for compiled languages
  - Subclass is dependent of base class (superclass)
  - Later changes to base class may damage subclass if not recompiled
  - The size of objects (i.e. structs or classes)
  - The offsets to "visible" (public or protected) data
  - The offsets to the virtual functions in the vtable
- Instead of inheritance use Containment/Delegation or Aggregation
- Please note: In .NET one of the strong points is the possibility to inherit even across programming languages. But there are strong checks at load time.

Containment or Delegation

- Outer Object implements A methods and B methods (because interface A extends B)
- Outer object’s implementation of B methods just calls Inner object’s B methods
  
  ```
  void m(int x) { b.m(x); }
  ```
- Inner Object is not aware of the special relationship
- Inner Object can be considered as a helper object

Aggregation

- Outer object puts inner object’s method pointer in its own method table
- Problem: normally Inner object would return its own IUnknown pointer if QueryInterface is done on interface B
- Solution: When Outer object creates Inner object, it passes its own IUnknown pointer as ‘Controlling Unknown’ CoCreateInstance(CLSID_CStat, pOuter, ...
Inheritance and Delegation in Corba

- Delegation can be used when a programming language does not support multiple inheritance.

```java
class A { void m(); }
class B { void n(); }
class C {
    private A a;
    private B b;
    C(A a, B b) {
        this.a = a;
        this.b = b;
    }
    m() { a.m(); }
    n() { b.n(); }
    p() { ....... }
}
```

Inheritance problem

- The problem with this solution is that C cannot be used where an A or B is required.
- A solution is to use interfaces and implementation classes.

```java
interface A { ... }
interface B { ... }
class A_impl implements A { ... }
class B_impl implements B { ... }
class C implements A, B {
    private A_impl a; // or private A a;
    private B_impl b; // or private B b;
    ... etc ...
}
```

or even interface C and class C_impl.

Corba Delegation Model (Tie)

- Servant class doesn’t have to extend skeleton.
- Only implement MyInterfaceOperations.
- Servant can inherit from another class.
- IDL compiler generates Tie class that inherits from skeleton.
- Tie class delegates to implementation class.
- Tie+implementation = “actual servant.”

```java
public class HelloPOATie extends HelloPOA {
    public HelloPOATie(HelloOperations delegate) {
        this._impl = delegate;
    }
    ...
    public String sayHello ()
    {
        return _impl.sayHello(); // delegation
    } // sayHello
```
Corba Delegation Model (Tie)

```java
class HelloImpl extends SomeOtherClass
    implements HelloOperations
    ...
    method implementations as usual

// Server class: create a servant.
    HelloImpl helloImpl = new HelloImpl();
// create a tie, with servant being the delegate.
    HelloPOATie tie = new HelloPOATie(helloImpl);
// obtain the objectRef for the tie
// this step also implicitly activates the object
    Hello href = tie._this(orb);
```

COM

- Dynamic Invocation
- Containment/Aggregation/Delegation
- Automation
- Monikers

(OLE) Automation

- Automation (formerly known as OLE Automation) is a way to script a program from another program
- E.g. Microsoft Word: Open a document, insert some text and save it.
- Usually done from scripting languages, e.g. Visual Basic, Python, Ruby.
- Application must make its operations available as (Automation) COM interfaces (IDispatch).
- Collection of interfaces known as OLE.
- For use in web browsers a stripped down collection is ActiveX.

Visual Basic and COM

- Visual Basic (up to version 6)
  - Developed as COM programming language
  - Seamless integration with COM

```vbnet
Dim statvar as Stat.Stat
Dim statvar as Object
Set statvar = CreateObject("Stat")
Set statvar = New Stat.Stat
statvar.init
statvar.AddItem 3.14
Set result = statvar.GetAvg
```

- When declared as Object uses runtime resolution
- When declared as specific class uses compile time resolution
- Type info comes from Type Library (loaded at compile time or runtime).

Automation with Visual Basic

```vbnet
Dim Word As Object
Dim Doc As Object
Dim Spot As Object
Set Word = CreateObject("Word.Application")
Word.Visible = True
Set Doc = Word.Documents.Add()
Set Spot = Doc.Range(0, 0)
Spot.InsertBefore "Hello From Visual Basic"
Doc.SaveAs "test.doc"
Doc.Close
Word.Quit
```

Early/Late Binding

- VB looks up the type info from the type library
- The Windows registry contains mapping from Program Id ("Word.Application") to Class Id
- The registry contains the location of the type library
- Variables declared as Object (Dim Word As Object) must be resolved at runtime
- This is called Late Binding.
- VB can also do the lookup at compile time: Early Binding:
  - Register the type library at compile time
  - Use: Dim Word As New Word.Application
Automation with Python

Can be used with ActiveState Python, or Standard Python + win32all extensions

```python
from win32com.client import Dispatch
word = Dispatch("Word.Application")
word.Visible = True
doc = word.Documents.Add()
spot = doc.Range(0, 0)
spot.InsertBefore("Hello from Python")
doc.SaveAs("test.doc")
doc.Close()
word.Quit()
```

Python/COM observations

- Python is very simple to use for Automation
- Python looks up the type library and makes all decisions at runtime
- You can even do it interactively
- This makes development and try-out very easy
- You can also run the makepy script
- This generates additional Python files containing the methods of the interface (like stubs)
- At runtime these are used if available
- This is comparable to Early Binding (faster)

Java and (D)COM

- Originally with Microsoft Java Virtual Machine (now defunct)
- Visual J++ 6.0 or MS Java SDK
- Almost transparent
- Java virtual machine does all the administration
- Build Interface pointers, refcounts etc.
- Special language extension is used (in specially formatted comments)
- Jactivex program generates interface and class files from TLB (Type Library)
- Wrapper classes: CCW (Com callable wrapper) to call Java from COM, JCW (Java callable wrapper) to call COM from Java
- J-Integra and EZ JCom are commercial implementations of the COM protocol in Java (any Java).

J-Integra example

```python
import excel.*;
...
Application app = new Application();
app.setVisible(true);
Workbooks workbooks = app.getWorkbooks();
Workbook workbook = workbooks.add(null);
Sheets worksheets = workbook.getWorksheets();
Worksheet sheet = new Worksheet(worksheets.add(null,
null, null, null));
Range range = sheet.getRange("A1:C3", null);
Object[][] newValue = {
{"abcd", new Boolean(false), new Double(98.0/12)},
{new Date(), new Integer(5454), new Float(22.0/7)},
{new Boolean(true), "xyzt", new Date() }
};
range.setValue(newValue); // Update the spreadsheet
```

Python – OpenOffice.org automation

Python as a separate process:

- Script programmer explicitly writes application startup code. Startup code must connect to Ooo. Code executes slow, as every call to Ooo is going through the inter process bridge.

Python embedded in OpenOffice.org process:

- Ooo listens on an inter process resource (socket or named pipe). This must be switched on seperately either on command line or with the Ooo configuration.

- Oo Process
- PyUNO Runtime
- Python Code
- Python process must be started separately.

- Oo Process
- PyUNO Runtime
- Python UNO component
- Script must be written as a PyUNO component (adding a coding overhead)
Py-UNO example

```python
import uno

# get the uno component context from the PyUNO runtime
localContext = uno.getComponentContext()

# create the UnoUrlResolver
resolver = localContext.ServiceManager.createInstanceWithContext(
    "com.sun.star.bridge.UnoUrlResolver",
    localContext )

# connect to the running office
text = model.Text
```
**Moniker use**

1. Client calls `IMoniker::BindToObject(IID_A)`
2. Moniker instantiates the object and tells it to load its persistent data
3. Pointer to `IMoniker` returned via moniker
4. Client invokes methods

**Moniker Examples**

- **File Moniker**: contents is a file, persistent data is file name
- **Item Moniker**: describes an item in a composite object, e.g. worksheet in an Excel workbook, persistent data is the item’s name and the moniker of the object in which it is contained
- **Composite Moniker**: groups together other monikers, e.g. File Moniker, worksheet item and range item. Persistent data: the component monikers
- **Java Moniker**: describes a Java class (now defunct):

  ```vbscript
  Dim dat As Object
  Set dat = CreateObject("java:java.util.Date")
  Dim s As String
  s = dat.toString
  ```

**Composite Moniker**

- **Word**
- **Composite Moniker**
  - File moniker: `Grades.xls`
  - Item moniker: `Sheet1`
  - Item moniker: `R1C1:R5C4`

**COM Structured Storage**

- **Structured Storage** is used for compound documents
- These documents can contain other (sub)documents
- E.g. a Word document containing an Excel sheet
- Each subdocument can be accessed more or less independently
- COM structured documents work like a filesystem within a file
- It has streams (file-like) and containers (directory-like)
- Interface IStream for file-like objects
  - Read, write, seek ...
- Interface IStorage for the containers
  - CreateStream, OpenStream for Streams in the Storage
  - CreateStorage, OpenStorage for sub-storages
  - methods for renaming, destroying, copying and moving
  - Transaction support (Commit, Revert)
  - Setting the class of an object, getting information ...
- Other interfaces to deal with persistence

**Structured Storage in general**

- Apple and IBM worked together on OpenDoc
- compound document architecture
- document consists of parts
- different parts can have different part editors
- part editors work in a part of the visible window
- parts can have different presentations
- parts can have different versions
- Now abandoned (sources are available)

---

**Example**

![Example Image](slides8.pdf)
Different representations

- Part can have different presentations
  - Presentation data
  - Can be stored with the frame

View type
- Determines the basic kind of presentation

What is the problem?

- Putting parts together in a single document
  - Reinventing the filesystem within a file
  - Better use the real filesystem
  - Or make mountable filesystems
  - No need to rewrite applications
  - Ship a document in a zip file

- Versioning
  - Use existing tools, like CVS, subversion
  - Contra: not integrated with compound documents

- Simultaneous access
  - Is usually done with database systems
  - Maybe file access should be upgraded to include transactions
  - Can be done with an additional Transactions layer