CS 242

Interoperability

John Mitchell

Topic

- Many systems built using multiple languages
  - Fortran calls to C code, vice versa
  - Microsoft: VBasic, C, C++
- Communication between modules
  - Function call with primitive-type arg and result?
  - Shared objects?
  - Error handling and exceptions?

Topic

- Many systems built using multiple languages
  - Fortran calls to C code, vice versa
  - Microsoft: VBasic, C, C++
- Communication between modules
  - Function call with primitive-type arg and result? Yes
  - Shared objects? Focus of this lecture
  - Error handling and exceptions? Still evolving

Three basic approaches

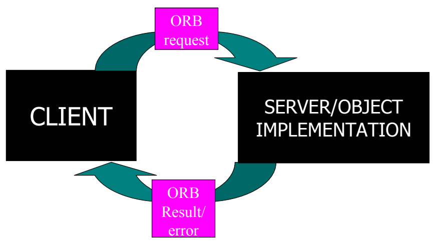
- Client-Implementation Intermediary
  - Convert messages from one implementation to another
  - Example: Corba
- Binary compatibility
  - Compatible values passed between implementations
  - Example: COM
- Neutral platform
  - Run on multiple languages on virtual machine
  - Example: .Net

Background: some related issues

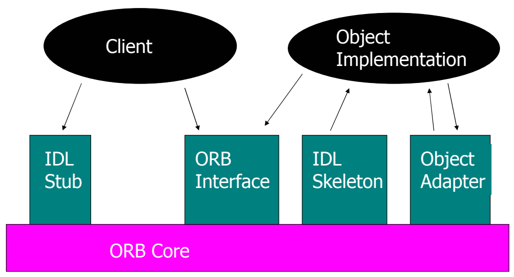
- Dynamic linking
  - Update parts of a system independently
    - C++: Link compiled code to stub
    - Stub contains code to interact with DLL
  - Share components among different applications
- Implementation compatibility
  - Can C++ component compiled with one compiler be dynamically linked with component from another?
- Inter-language interoperability
  - Two languages ~ two compilers

Corba Concept

- Insert "broker" between client and server



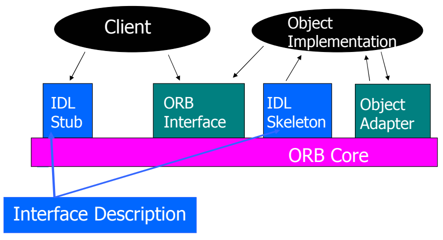
Corba Architecture



Functions of ORB

- ◆Communication between client and server
  - Insulates application system configuration details
- ◆Specific steps
  - Intercepts calls
  - Finds object
  - Invokes method
  - Passes parameters
  - Returns results or error messages

Interface description language

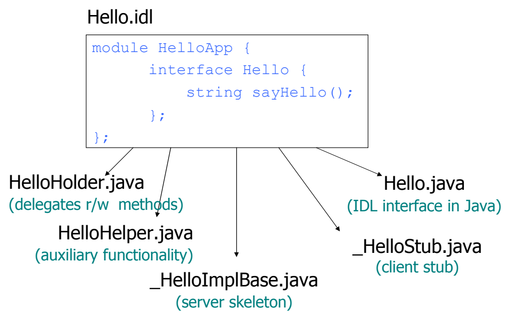


- IDL generates 'stubs' and 'skeleton' programs for each interface
- 'Stub' acts like a local function call, providing interface to ORB
- 'Skeleton' yields server side implementation of IDL interface

CORBA application development

- ◆Write the IDL interface
- ◆Map .idl file to target language (C++, Java, …)
  - IDL Compiler
- ◆Develop a Client application
- ◆Develop the Server
- ◆Compile and run the application

IDL Example

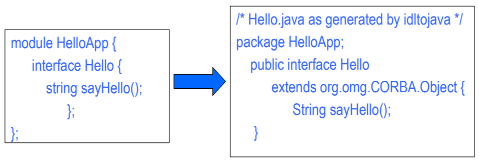


Hello.idl

```
module HelloApp {
      interface Hello {
          string sayHello();
      };
};
```

Interface translation

◆IDL Statement	Java Statement
module HelloApp	package HelloApp;
interface Hello	public interface Hello
string sayHello();	String sayHello();



```
module HelloApp {
    interface Hello {
        string sayHello();
    };
};
```

```
/* Hello.java as generated by idltojava */
package HelloApp;
    public interface Hello
        extends org.omg.CORBA.Object {
            String sayHello();
    }
```

IDL Compiler Output

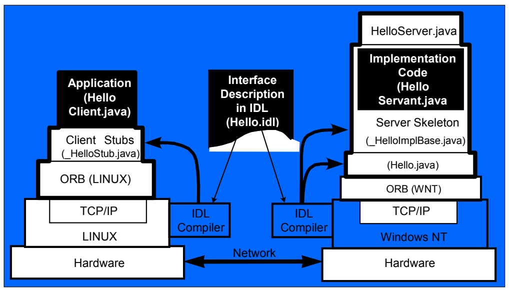
- _HelloImplBase.java
  - Abstract class provides the server skeleton, basic CORBA functionality
  - Implements Hello.java interface
  - server class HelloServant extends _HelloImplBase
- _HelloStub.java
  - Client stub, with CORBA functionality for the client
  - Implements Hello.java interface.
- Hello.java
  - Interface containing Java version of IDL interface, extends org.omg.CORBA.Object
- HelloHelper.java
  - Final class provides auxiliary functionality, including a narrow method to cast CORBA object references to proper type
- HelloHolder.java
  - Final class has public instance member of type Hello
  - Provides operations for out and in/out arguments, which CORBA assumes but do not map immediately to Java

Just to give you some idea of what this looks like ...

Implementing the Client

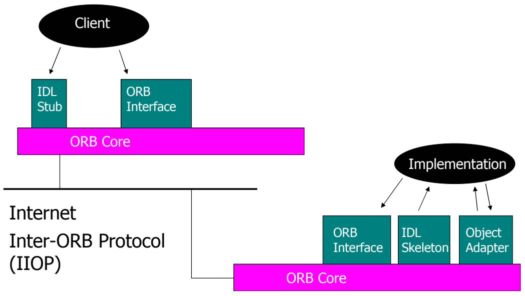
```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient {
  static Hello helloImpl;
  public static void main(String args[]) {
    try{     // create and initialize the ORB
      ORB orb = ORB.init(args, null);
      // get the root naming context
      org.omg.CORBA.Object objRef =  orb.resolve_initial_references("NameService");
      // Use NamingContextExt instead of NamingContext; part of Interoperable naming Service
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      // resolve the Object Reference in Naming
     String name = "Hello";
     helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
     System.out.println("Obtained a handle on server object: " + helloImpl);
     System.out.println(helloImpl.sayHello());
     helloImpl.shutdown();
    }
catch (Exception e) {   System.out.println("ERROR : " + e) ;   e.printStackTrace(System.out);
} } }
```

Hello World Architecture

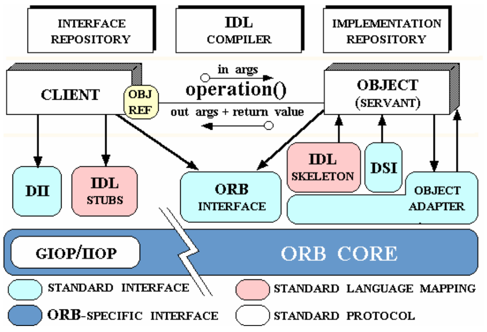

HelloServer.java
Implementation Code (Hello Servant.java
Server Skeleton (_HelloImplBase.java)
(Hello.java)
ORB (WNT)
TCP/IP
Windows NT
Hardware
Application (Hello Client.java)
Client Stubs (_HelloStub.java)
ORB (LINUX)
TCP/IP
LINUX
Hardware
Interface Description in IDL (Hello.idl)
IDL Compiler
IDL Compiler
Network

See http://java.sun.com/j2se/1.4.2/docs/guide/idl/jidlExample.html

Inter-ORB Communication



Corba ORB Architecture: more details



Corba Summary

- Interface definition language (IDL)
  - Define interface in "neutral language"
  - Compiler generates several related files automatically
- Object request broker (ORB)
  - System intermediary handles requests, response

COM: Component Object Model

- Purpose (marketing page)
  - "COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. …"
- Current incarnations
  - COM+, Distributed COM (DCOM) , ActiveX Controls
- References
  - Don Box, Essential COM
  - MS site: http://www.microsoft.com/com/

Central ideas in COM

- Clients program using interfaces, not classes
- Implementation code is dynamically linked
- Manage requirements at run time
  - Object implementors declare runtime requirements
  - System ensures that these requirements are met
- Evolution
  - Interfaces and dynamic linking are classic COM
  - Runtime requirements handled using Microsoft Transaction Server (MTS) and COM+

Motivation

- Build dynamically composable systems
  - Not all parts of application are statically linked
- Independence of components
  - One change should not propagate to all source code
  - New components usable in place of old
- Compatibility of components
  - Use components with different runtime requirements
  - Mix heterogeneous objects in a single process

Evolution in appreciation for abstraction

- 1980s: classes and objects
  - Classes used for object implementation
  - Classes also used for consumer/client type hierarchy
- Dependencies between client and object
  - Client assumes complete knowledge of public interface
  - Client may know even more under certain languages (C++)
- 1990s: separate interface from implementation
  - Client to program in terms of abstract types
  - Completely hides implementation class from client

Interface-Based Programming

- Define interfaces for classes
  - C++ : use abstract base classes as interfaces
- Associate implementations with interfaces
  - C++: inheritance
    - Class FastString : public IFastString {...};
- Create implementation objects without exposing layout
  - Usually a creator or factory function
  - Manipulate object indirectly through intermediate structure
  - Class unsuitable for declaring variables
    - Want to avoid dependence on class
- Client must be able delete object
  - Since new operator not used by the client, cannot call delete
  - Reference counting can be used

Interoperability

- Use of interfaces separates implementation
  - Different implementations can coexist
  - These can be built in different languages

Interfaces in different languages

- Pure abstract base classes in C++
  - All methods are pure virtual
  - Never any code, only signature
  - Format of C++ vtable/vptr defines expected stack frame
- Represented directly as interfaces in Java
- Represented as Non-Creatable classes in Visual Basic
- Uniform binary representation independent of how object is built
- Identified uniquely by a 128-bit Interface ID (IID)

IUnknown

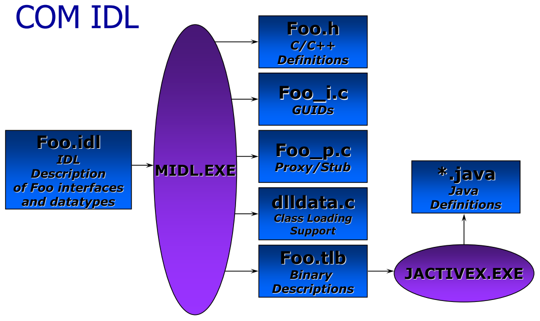
- Three abstract operations (QueryInterface, AddRef, and Release) comprise the core interface of COM, IUnknown
- All COM interfaces must extend IUnknown
- All COM objects must implement IUnknown

```
extern const IID IID_IUnknown;
struct IUnknown {
  virtual HRESULT STDMETHODCALLTYPE QueryInterface(
                 const IID& riid, void **ppv) = 0;
  virtual ULONG STDMETHODCALLTYPE AddRef( ) = 0;
  virtual ULONG STDMETHODCALLTYPE Release( ) = 0;
};
```

IDL: generate Interfaces

- COM interfaces may be defined in COM IDL
- IDL compiler generates C/C++ interface definitions

COM IDL



- COM interfaces may be defined in COM IDL
- IDL compiler generates C/C++ interface definitions

COM IDL

- All elements in an IDL file can have attributes
  - Appear in [ ] prior to subject of attributes
- Interfaces are defined at global scope
  - Required by MIDL to emit networking code
- Must refer to exported types inside library block
  - Required by MIDL to emit type library definition
- Can import std interface suite
  - WTYPES.IDL - basic data types
  - UNKNWN.IDL - core type interfaces
  - OBJIDL.IDL - core infrastructure itfs
  - OLEIDL.IDL - OLE itfs
  - OAIDL.IDL - Automation itfs
  - OCIDL.IDL - ActiveX Control itfs

COM IDL

CalcTypes.idl

```
[ uuid(DEFACED1-0229-2552-1D11-ABBADABBAD00), object ]
interface ICalculator : IDesktopDevice {
    import "dd.idl"; // bring in IDesktopDevice
    HRESULT Clear(void);
    HRESULT Add([in] short n);       //   n sent to object
    HRESULT GetSum([out] short *pn); // *pn sent to caller
}
[
  uuid(DEFACED2-0229-2552-1D11-ABBADABBAD00),
  helpstring("My Datatypes")
]
library CalcTypes {
  importlib("stdole32.tlb"); // required
  interface ICalculator;     // cause TLB inclusion
}
```

COM IDL - C++ Mapping

CalcTypes.h

```
#include “dd.h”
extern const IID IID_ICalculator;
struct
__declspec(uuid(“DEFACED1-0229-2552-1D11-ABBADABBAD00”))
ICalculator : public IDesktopDevice {
   virtual HRESULT STDMETHODCALLTYPE Clear(void) = 0;
   virtual HRESULT STDMETHODCALLTYPE Add(short n) = 0;
   virtual HRESULT STDMETHODCALLTYPE GetSum(short *pn) = 0;
};
extern const GUID LIBID_CalcTypes;
```

CalcTypes_i.c

```
const IID IID_ICalculator = {0xDEFACED1, 0x0229, 0x2552,
 { 0x1D, 0x11, 0xAB, 0xBA, 0xDA, 0xBB, 0xAD, 0x00 } };
const GUID LIBID_CalcTypes = {0xDEFACED2, 0x0229, 0x2552,
 { 0x1D, 0x11, 0xAB, 0xBA, 0xDA, 0xBB, 0xAD, 0x00 } };
```

COM IDL – Java/VB Mapping

CalcTypes.java

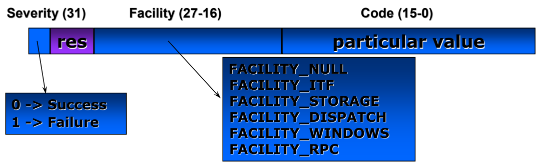
```
package CalcTypes; // library name
/**@com.interface(iid=DEFACED1-0229-2552-1D11-ABBADABBAD00)*/
interface ICalculator extends IDesktopDevice {
    public void Clear( );
    public void Add(short n);
    public void GetSum(short [] pn); // array of length 1
    public static com.ms.com._Guid iid =
      new com.ms.com._Guid(0xDEFACED1, 0x0229, 0x2552,
                           0x1D, 0x11, 0xAB, 0xBA,
                           0xDA, 0xBB, 0xAD, 0x00);
}
```

CalcTypes.cls

```
Public Sub Clear( )
Public Sub Add(ByVal n As Integer)
Public Sub GetSum(ByRef pn As Integer)
```

COM And Error Handling

- COM (today) doesn’t support typed C++ or Java-style exceptions
- All (remotable) methods must return a standard 32-bit error code called an HRESULT
  - Mapped to exception in higher-level languages
  - Overloaded to indicate invocation errors from proxies



COM Data Types

IDL	C++	Java	Visual Basic
small	char	byte	N/A
short	short	short	Integer
long	long	int	Long
hyper	__int64	long	N/A
unsigned small	unsigned char	byte	*Byte*
unsigned short	unsigned short	short	N/A
unsigned long	unsigned long	int	N/A
unsigned hyper	unsigned __int64	long	N/A
float	float	float	Single
double	double	double	Double
char	char	char	N/A
unsigned char	unsigned char	byte	Byte
wchar_t	wchar_t	char	*Integer*

COM Data Types

IDL	C++	Java	Visual Basic
byte	unsigned char	*char*	N/A
BYTE	unsigned char	byte	Byte
boolean	long	int	Long
VARIANT_BOOL	VARIANT_BOOL	boolean	Boolean
BSTR	BSTR	java.lang.String	String
VARIANT	VARIANT	com.ms.com.Variant	Variant
CY	long	int	Currency
DATE	double	double	Date
enum	enum	int	Enum
Typed ObjRef	IFoo *	interface IFoo	IFoo
struct	struct	final class	Type
union	union	N/A	N/A
C-style Array	array	array	N/A

Reference counting

- Leverage indirection through reference object
  - Clients “Delete” each reference, not each object
- Object checks references to it
  - Objects track number of references and auto-delete when count reaches zero
  - Requires 100% compliance with ref. counting rules
- Client obligations
  - All operations that return interface pointers must increment the interface pointer’s reference count
  - Clients must inform object that a particular interface pointer has been destroyed

COM Summary

- Clients program using abstract data types: interfaces
- Clients can load method code dynamically without concern for C++ compiler incompatibilities
- Clients interrogate objects for extended functionality via RTTI-like constructs
- Clients notify objects when references are duplicated or destroyed
- Supports multi-language programming

.NET Framework

- Microsoft cross-language platform
  - Many languages can use and extend .NET Framework
    - Compile language to MSIL
  - All languages are interoperable
- Focus on security and trust
  - Building, deploy and run semi-trusted applications
- Two key components
  - Common Language Runtime
  - .NET Framework Class Library

Slide credit: Graphics, etc. stolen from MS slides on web

Current .NET Languages

- C++
- Visual Basic
- C#
- Jscript
- J#
- Perl
- Python
- Fortran
- COBOL
- Eiffel
- Haskell
- SmallTalk
- Oberon
- Scheme
- Mercury
- Oz
- RPG
- Ada
- APL
- Pascal
- ML

Language Examples

J#

```
String s = "authors";
SqlCommand cmd = new SqlCommand("select * from "+s, sqlconn);
cmd.ExecuteReader();
```

VB.NET

```
Dim s as String
s = "authors"
Dim cmd As New SqlCommand("select * from " & s, sqlconn)
cmd.ExecuteReader()
```

Language Examples

C#

```
string s = "authors";
SqlCommand cmd = new SqlCommand("select * from "+s, sqlconn);
cmd.ExecuteReader();
```

C++

```
String *s = S"authors";
SqlCommand cmd = new
SqlCommand(String::Concat(S"select * from ", s),
                     sqlconn);
cmd.ExecuteReader();
```

Language Examples

JScript

```
var s = "authors"
var cmd = new SqlCommand("select * from " + s, sqlconn)
cmd.ExecuteReader()
```

Perl

```
String *s = S"authors";
SqlCommand cmd = new SqlCommand(String::Concat(S"select *
from ", s), sqlconn);
cmd.ExecuteReader();
```

Python

```
s = "authors"
cmd =SqlCommand("select * from " + s, sqlconn)
cmd.ExecuteReader()
```

Language Example

Cobol

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
    CLASS SqlCommand AS "System.Data.SqlClient.SqlCommand"
    CLASS SqlConnection AS "System.Data.SqlClient.SqlConnection".
DATA DIVISION.
WORKING-STORAGE SECTION.
01 str PIC X(50).
01 cmd-string PIC X(50).
01 cmd OBJECT REFERENCE SqlCommand.
01 sqlconn OBJECT REFERENCE SqlConnection.
PROCEDURE DIVISION.
 *> Establish the SQL connection here somewhere.
MOVE "authors" TO str.
STRING "select * from " DELIMITED BY SIZE,
   str DELIMITED BY " " INTO cmd-string.
INVOKE SqlCommand "NEW" USING BY VALUE cmd-string sqlconn RETURNING cmd.
INVOKE cmd "ExecuteReader".
```

Language Examples

RPG

```
DclFld MyInstObj Type( System.Data.SqlClient.SqlCommand )
DclFld s Type( *string )
s = "authors"
MyInstObj = New System.Data.SqlClient.SqlCommand("select *
                from "+s, sqlconn)
MyInstObj.ExecuteReader()
```

Fortran

```
assembly_external(name="System.Data.SqlClient.SqlCommand")
sqlcmdcharacter*10 xsqlcmd
Cmd x='authors'
cmd = sqlcmd("select * from "//x, sqlconn)
call cmd.ExecuteReader()
end
```

Language Examples

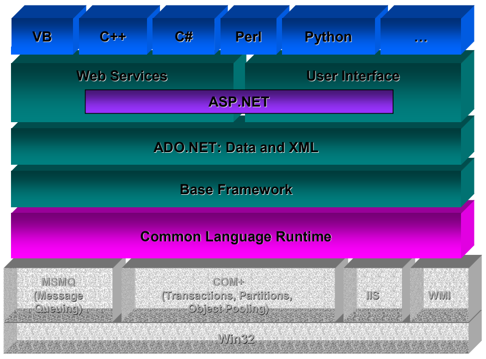
Scheme

```
(let* (  (s "authors")
  (cmd (new-SqlCommand (string-append "select * from " s)
sqlconn)))
(execute-command cmd))
```

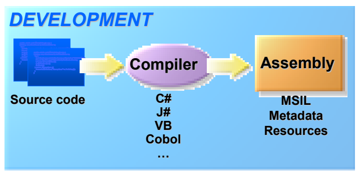
Eiffel

```
local
      s: STRING
      cmd: SQLCOMMAND
do
      s := "authors"
      create cmd("select * from " + s, sqlconn)
      cmd.ExecuteReader()
end
```

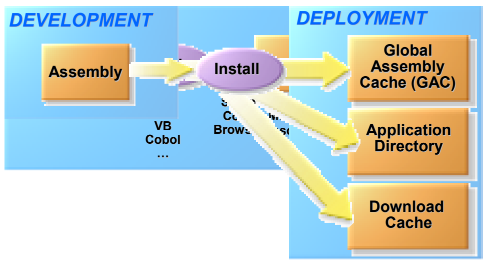
Framework Architecture



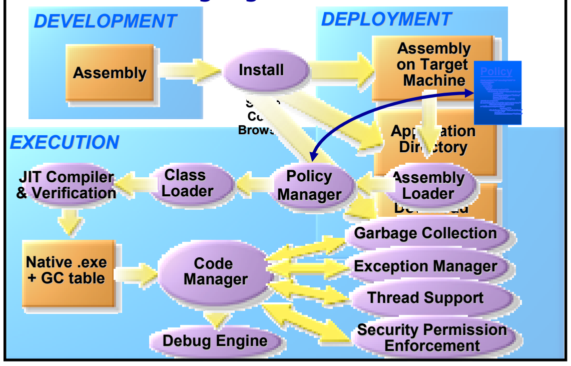
Common Language Runtime



Common Language Runtime



Common Language Runtime



Security Issues

- ◆OS security is based on user rights
- ◆CLR security, added on top of OS security, gives rights to code

! Trusted user Untrusted code	Trusted user Trusted code
Untrusted user Untrusted code	! Untrusted user Trusted code

.NET Summary

- ◆Compile multiple languages to common intermediate language (MSIL)
- ◆MSIL executed by virtual machine
  - Similar to Java VM in many respects
  - More elaborate security model
  - JIT is standard, instead of interpreter
- ◆MSIL contains special provisions for certain languages

Summary

- ◆CORBA
  - Interoperability through programming conventions, ORB intermediary
- ◆COM
  - Conventions for producing binary-compatible objects
  - Client uses interface only, no knowledge of object format
- ◆.NET
  - Compile multiple language to single MSIL
  - MSIL handles dynamic linking/loading, security, …
    - "managed" and "unmanaged" code