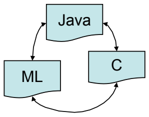
Interoperability

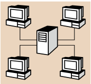
Kathleen Fisher

Why is interoperability important?

- Write each part of a complex system in a language well-suited to the task:
  - C for low-level machine management
  - Java/C#/Objective-C for user-interface
  - Ocaml/ML for tree transformations



- Integrate existing systems:
  - implemented in different languages
  - for different operating systems
  - on different underlying hardware systems

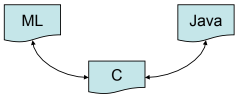


Why is it hard?

- Languages make different choices:
  - Function calling conventions
    - caller vs callee saved registers
  - Data representations
    - strings, object layout
  - Memory management
    - tagging scheme
- Interoperating requires bridging the gap.

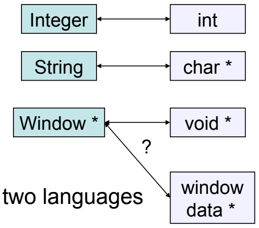
C/C++ as Lingua Franca

- Ubiquitous
- Computation model *is* underlying machine:
  - Other languages already understand.
  - No garbage collection.
- Representations well-known and fixed
  - Millions of lines of code would break if changed.



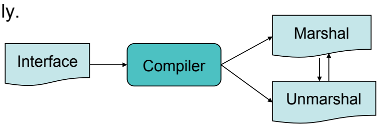
Marshaling and Unmarshaling

- Convert data representations from one language to another.
- Easier when one end is C as rep is known.
- Policy choice: copy or leave abstract?
- Tedious, low-level
- Modulo policy, fixed by two languages



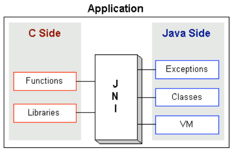
Interface specifications

- Contract describing what an implementation written in one language will provide for another.
  - Inferred from high-level language: JNI
  - Inferred from C header files: SWIG
  - Specified in Interface Definition Language: ocamlidl, COM, CORBA
- Allow tools to generate marshaling/unmarshaling code automatically.



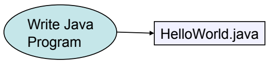
JNI: Integrating C/C++ and Java

- Java Native Interface
  - Allows Java methods to be implemented in C/C++.
  - Such methods can
    - create, inspect, and send messages to Java objects
    - modifiy Java objects & have changes reflected to system
    - catch and throw exceptions C that Java will handle.
- JNI enforces policy in which pointers are abstract.

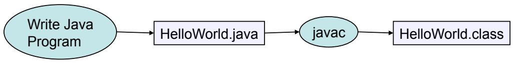


java.sun.com/docs/books/tutorial/native1.1/TOC.html

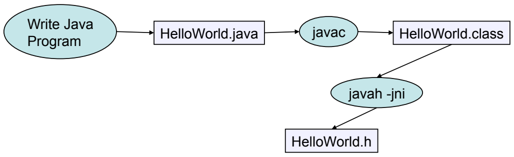
JNI Example: Hello World!



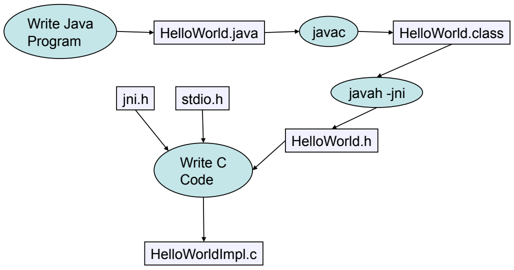
JNI Example: Hello World!



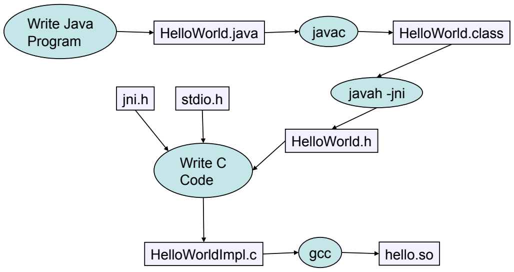
JNI Example: Hello World!



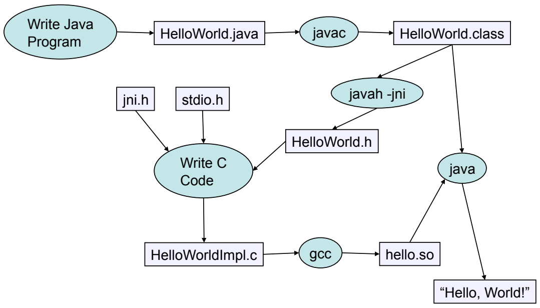
JNI Example: Hello World!



JNI Example: Hello World!

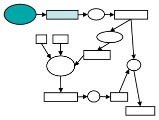


JNI Example: Hello World!


Write Java Program
HelloWorld.java
javac
HelloWorld.class
jni.h
stdio.h
javah -jni
HelloWorld.h
Write C Code
java
HelloWorldImpl.c
gcc
hello.so
"Hello, World!"

JNI Example: Write Java Code

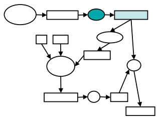
```
class HelloWorld {
  public native void displayHelloWorld();

  static {
        System.loadLibrary("hello");
  }

  public static void main(String[] args) {
        new HelloWorld().displayHelloWorld();
  }
}
```



JNI Example: Compile Java Code

`javac HelloWorld.java`

```
café babe 0000 002e 001b 0a00 0700 1207
0013 0a00 0200 120a 0002 0014 0800 130a
…
```

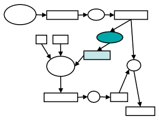


JNI Example: Generate C Header

```
javah –jni
HelloWorld.java
```

```
#include <jni.h>
/* Header for class HelloWorld */
#ifndef _Included_HelloWorld
 #define _Included_HelloWorld
 #ifdef __cplusplus
   extern "C" {
 #endif

JNIEXPORT void JNICALL
  Java_HelloWorld_displayHelloWorld
  (JNIEnv *, jobject);
#endif
```

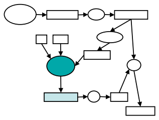
Function takes two "extra" arguments:
- environment pointer
- object pointer (*this*)



JNI Example: Write C Method

```
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>

JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld(JNIEnv *env, jobject obj) {
    printf("Hello world!\n");
    return;
}
```

Implementation includes 3 header files:
- jni.h: provides information that C needs to interact with JVM
- HelloWorld.h: generated in previous step
- stdio.h: provides access to `printf`.



JNI Example: Create Shared Lib

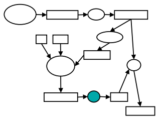
How to create a shared library depends on platform:

Solaris:

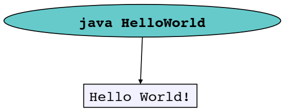
```
cc –G –I/usr/local/java/include \
      –I/usr/local/java/include/solaris \
      HelloWorldImp.c –o libhello.so
```

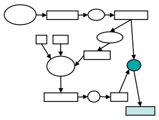
Microsoft Windows w/ Visual C++ 4.0:

```
cl –Ic:\java\include
   –Ic:\java\include\win32
   –LD HelloWorldImp.c –Fehello.dll
```



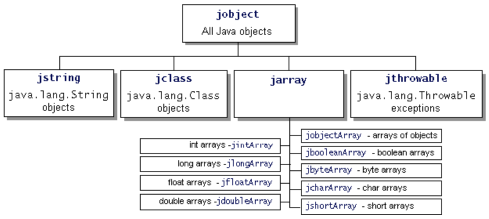
JNI Example: Run Program





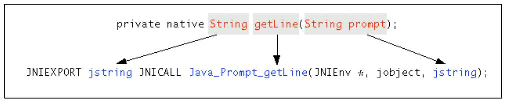
JNI: Type Mapping

- Java primitive types map to corresponding types in C.
- All Java object types are passed by reference.



JNI: Method Mapping

- The `javah` tool uses type mapping to generate prototypes for native methods:



```
private native String getLine(String prompt);

JNIEXPORT jstring JNICALL Java_Prompt_getLine(JNIEnv *, jobject, jstring);
```

JNI: Accessing Java Strings

- Type `jstring` is not `char *`!
- Native code must treat `jstring` as an abstract type and use `env` functions to manipulate:

```
JNIEXPORT jstring JNICALL
Java_Prompt_getLine(JNIEnv *env, jobject obj, jstring prompt)
{
    char buf[128];
    const char *str = (*env)->GetStringUTFChars(env, prompt, 0);
    printf("%s", str);
    (*env)->ReleaseStringUTFChars(env, prompt, str);
    ...
    scanf("%s", buf);
    return (*env)->NewStringUTF(env, buf);
}
```

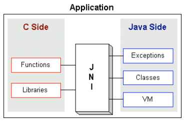
JNI: Calling Methods

- Native methods can invoke Java methods using the environment argument:

```
JNIEXPORT void JNICALL
Java_Callbacks_nativeMethod(JNIEnv *env, jobject obj, jint depth)
{
    jclass cls = (*env)->GetObjectClass(env, obj);
    jmethodID mid = (*env)->GetMethodID(env, cls, "callback", "(I)V");
    if (mid == 0) {
        return;
    }
    printf("In C, depth = %d, about to enter Java\n", depth);
    (*env)->CallVoidMethod(env, obj, mid, depth);
    printf("In C, depth = %d, back from Java\n", depth);
}
```

JNI: Summary

- Allows Java methods to be implemented in C/C++.
- Interface determined by native method signature.
- Tools generate C interfaces and marshaling code.
- References are treated abstractly, which facilitates memory management.
- Environment pointer provides to JVM services such creation and method



SWIG

- Tool to make C/C++ libraries easily available in many high level languages:

Tcl, Python, Perl, Guile, Java, Ruby, Mzscheme, PHP, Ocaml, Pike, C#, Allegro CL, Modula-3, Lua, Common Lisp, JavaScript, Eiffel, …

- Goal: Read interface from C/C++ headers, requiring annotations only to customize.
- Marshaling policy: references treated opaquely. C library must provide extra functions to allow high-level language to manipulate.

www.swig.org

Interface Definition Languages

- IDLs provide some control over marshaling policies:
  - Are parameters `in`, `out`, or both?
  - Is `NULL` a distinguished value?
  - Should payload of pointers be copied or left abstract?
  - Is char* a pointer to a character or a string?
  - Is one parameter the length of an argument array?
  - Who is responsible for allocating/deallocating space?
- Language-specific IDL compilers generate glue code for marshaling/unmarshaling.

IDLs

- Typically look like C/C++ header files with additional declarations and attributes.

```
int foo([out] long* l,
        [string, in] char* s,
        [in, out] double * d);
```

- Annotations tell high-level language how to interpret C/C++ parameters.
- Unlike SWIG, pointers don't have to be abstract on high-level language side.
- Unlike JNI, pointers don't have to be abstract on C side.

IDLs: Pointer Annotations

- Five annotations to clarify role of pointers:
  - `ref`: a unique pointer that can be safely marshaled.
  - `unique`: just like `ref` except it may also be `null`.
  - `ptr`: could be shared, could point to cyclic data; can't be marshaled.
  - `string char*`: null terminated sequence of characters, should be treated like a string.
  - `size_is(parameter_name)` : pointer is array whose length is given by `parameter_name`.

```
void DrawPolygon
     ([in, size_is(nPoints)] Point* points,
      [in] int nPoints)
```

Examples of IDL-based Systems

- Simple high-level language to C bindings:
  - camlidl, H/Direct, mlidl, etc.
- COM: Microsoft's interoperability platform.
- CORBA: OMG's interoperability platform.

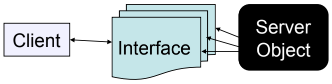
COM and CORBA both leverage the idea of IDLs to go well beyond simple interoperability, supporting distributed *components*: collections of related behaviors grouped into objects.

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. …"
- Used in applications like Microsoft Office.
- Current incarnations
  - COM+, Distributed COM (DCOM) , ActiveX Controls
- References
  - Don Box, Essential COM
  - MS site: http://www.microsoft.com/com/

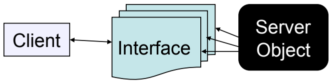
COM

- Each object (aka *server*) supports multiple interfaces, each representing a different view of the object.
- COM clients acquire pointers to one of an object's interfaces and invoke methods through that pointer as if object were local.
- All COM objects provide `QueryInterface` method to support dynamic interface discovery.



Versioning

- Microsoft uses multiple interfaces to support versioning.
- Objects never modify existing interfaces, merely add new ones.
- New client code asks for newer server interfaces; legacy code can continue to ask for older versions.

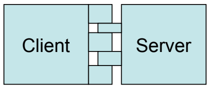


Binary Compatibility

- COM specifies that object implementations must conform to C++ vtable layout.
- Each object can be implemented in any language as long as compiler for language can produce vtables.
- Interfaces of COM objects described in MIDL.
- Language-specific IDL compiler generates proxy/stub functions for marshaling and unmarshaling to a wire format.

Execution Model, Local

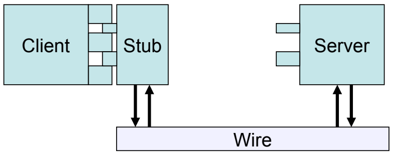
- If executing in the same address space, client and server objects are dynamically linked.



- The first time a message is sent to server, code in initial stub vtable finds and loads code, replacing itself with the actual vtable.

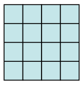
Execution Model, Remote

- If executing in different address spaces, stub vtable marshals arguments, sends message to remote object, waits for response, unmarshals it and delivers it.



COM: Grid Example

- Grid server object maintains two dimensional array of integers.
- Supports two groups of methods:



IGrid1

get() : gets value stored at argument location.
set() : sets value at argument location.

IGrid2

reset() : resets value of all cells to supplied value.

COM: Grid Example IDL

- Portion of IDL file to describe IGrid1 interface:

```
// uuid and definition of IGrid1
[  object,
   uuid(3CFDB283-CCC5–11D0-BA0B-00A0C90DF8BC),
   helpstring("IGrid1 Interface"),
   pointer_default(unique)
]
interface IGrid1 : IUnknown      {
   import "unknwn.idl";
   HRESULT get([in] SHORT n, [in] SHORT m, [out] LONG *value);
   HRESULT set([in] SHORT n, [in] SHORT m, [in] LONG value);
};
```

- Each interface has a globally unique GUID and extends the `IUnknown` interface, which provides `queryInterface` and reference counting methods.

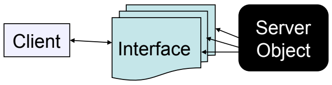
COM: Grid Example Client Code

```
#include "grid.h”
void main(int argc, char**argv) {
    IGrid1 *pIGrid1;
    IGrid2 *pIGrid2;
    LONG    value;
    CoInitialize(NULL);            // initialize COM
    CoCreateInstance(CLSID_CGrid, NULL, CLSCTX_SERVER,
                     IID_IGrid1, (void**) &pIGrid1);
    pIGrid1->get(0, 0, &value);
    pIGrid1->QueryInterface(IID_IGrid2, (void**) &pIGrid2);
    pIGrid1->Release();
    pIGrid2->reset(value+1);
    pIGrid2->Release();
    CoUninitialize();
}
```

my.execpc.com/~gopalan/misc/compare.html

COM Summary

- Object servers are abstract data types described by interfaces.
- Object servers can be loaded dynamically and accessed remotely.
- Clients interrogate server objects for functionality via RTTI-like constructs (ie, `queryInterface`).
- Clients notify server objects when references are duplicated or destroyed to manage memory.
- Supports binary-compatible multi-language programming.



CORBA

- Interoperability where systems can’t be tightly coupled:
  - Companies working together (telecommuni medical, etc.)
  - Large system integrations

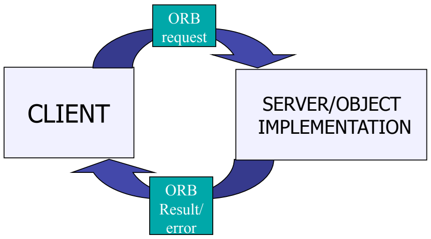
- Can’t enforce same language, same OS, or same hardware.
  - Engineering tradeoffs, cost effectiveness, legacy systems

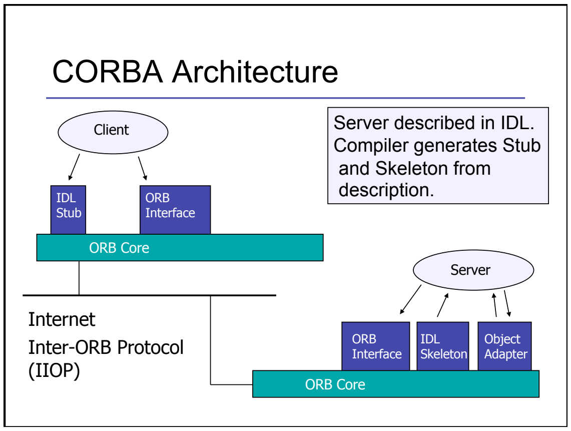
OMG

- CORBA is a standard developed by the Object Management Group.
  - Common Object Request Broker Architecture
  - Over 700 participating companies
  - Request for proposal process
- Example:
  - Telecommunications industry uses CORBA to manage provisioning process, in which competitors have to work together.

CORBA Concept

- Insert “broker” between client and server, called the Object Request Broker.

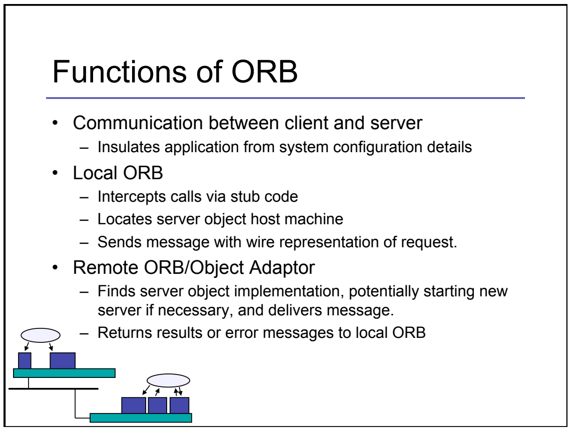




CORBA Architecture

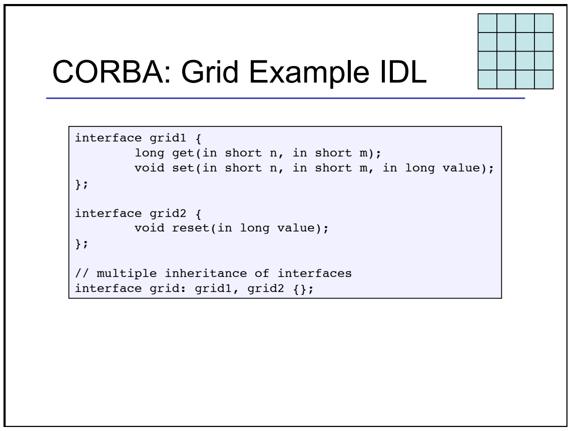
Server described in IDL. Compiler generates Stub and Skeleton from description.

Internet
Inter-ORB Protocol
(IIOP)



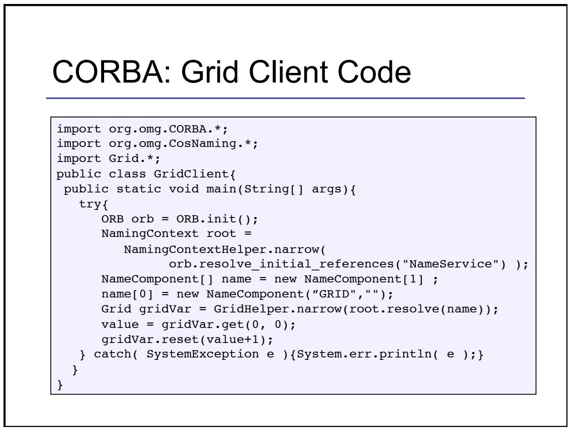
Functions of ORB

- Communication between client and server
  - Insulates application from system configuration details
- Local ORB
  - Intercepts calls via stub code
  - Locates server object host machine
  - Sends message with wire representation of request.
- Remote ORB/Object Adaptor
  - Finds server object implementation, potentially starting new server if necessary, and delivers message.
  - Returns results or error messages to local ORB



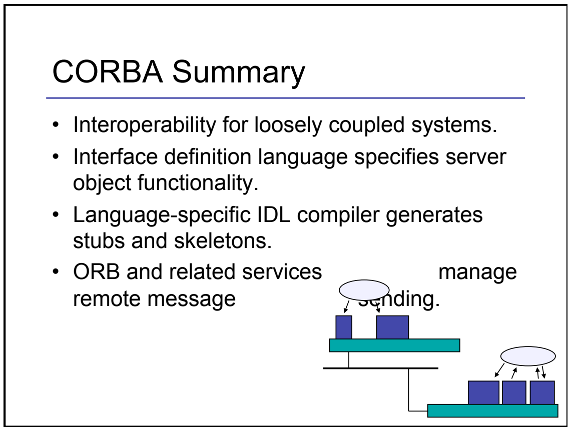
CORBA: Grid Example IDL

```
interface grid1 {
        long get(in short n, in short m);
        void set(in short n, in short m, in long value);
};

interface grid2 {
        void reset(in long value);
};

// multiple inheritance of interfaces
interface grid: grid1, grid2 {};
```



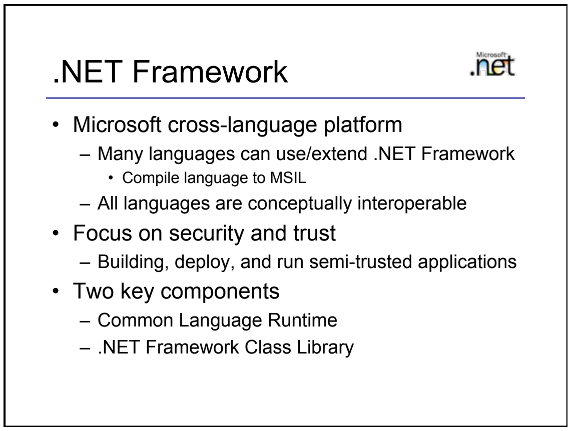
CORBA: Grid Client Code

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import Grid.*;
public class GridClient{
 public static void main(String[] args){
   try{
      ORB orb = ORB.init();
      NamingContext root =
         NamingContextHelper.narrow(
               orb.resolve_initial_references("NameService") );
      NameComponent[] name = new NameComponent[1] ;
      name[0] = new NameComponent(”GRID","");
      Grid gridVar = GridHelper.narrow(root.resolve(name));
      value = gridVar.get(0, 0);
      gridVar.reset(value+1);
   } catch( SystemException e ){System.err.println( e );}
  }
}
```



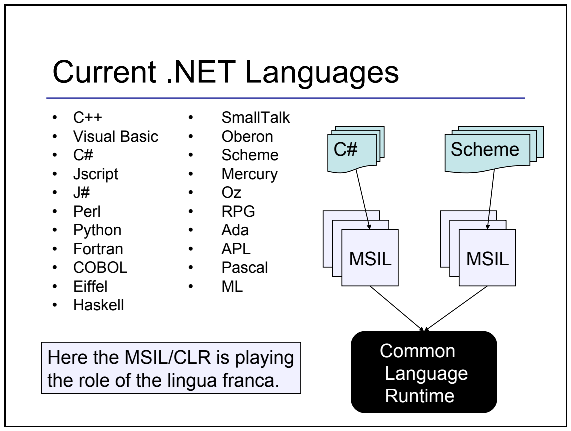
CORBA Summary

- Interoperability for loosely coupled systems.
- Interface definition language specifies server object functionality.
- Language-specific IDL compiler generates stubs and skeletons.
- ORB and related services manage remote message sending.



.NET Framework

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



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

C#

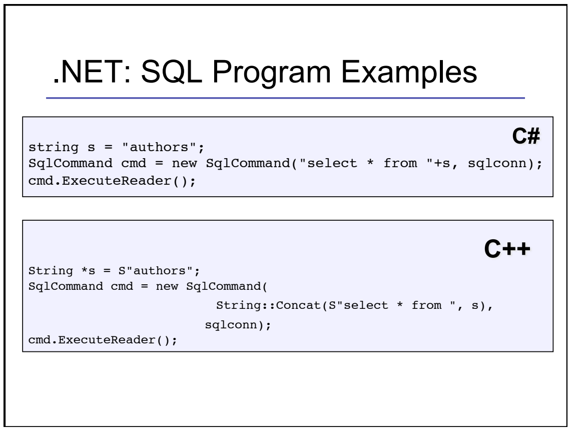
Scheme

MSIL

MSIL

Common Language Runtime

Here the MSIL/CLR is playing the role of the lingua franca.



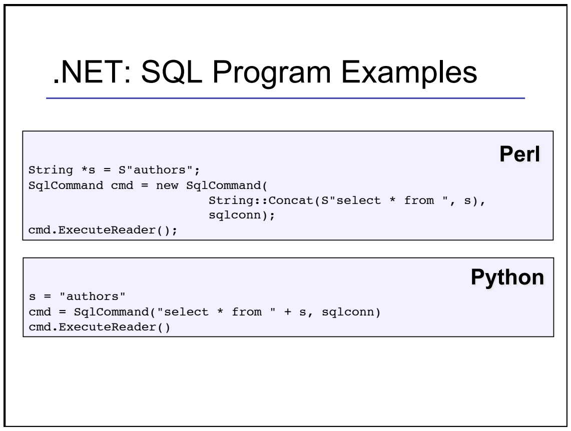
.NET: SQL Program Examples

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();
```



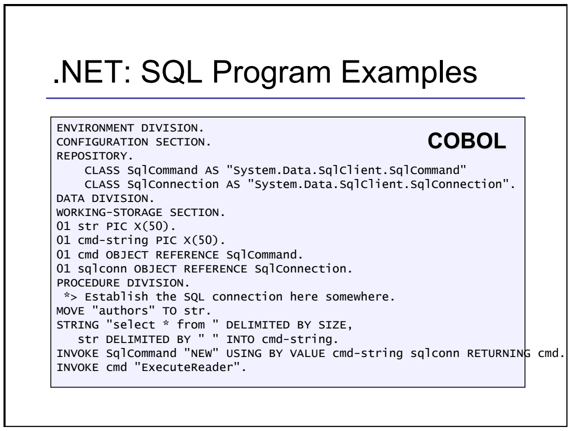
.NET: SQL Program Examples

Perl

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

Python

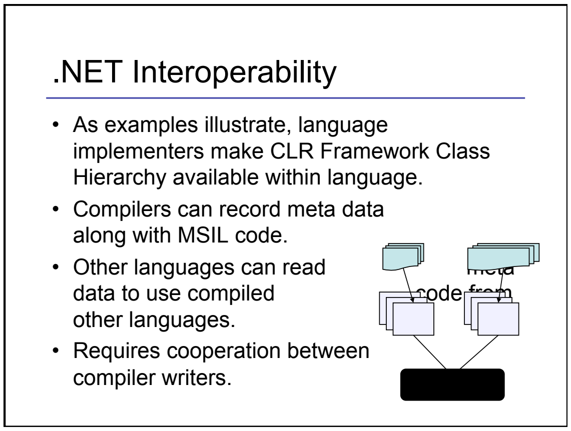
```
s = "authors"
cmd = SqlCommand("select * from " + s, sqlconn)
cmd.ExecuteReader()
```



.NET: SQL Program Examples

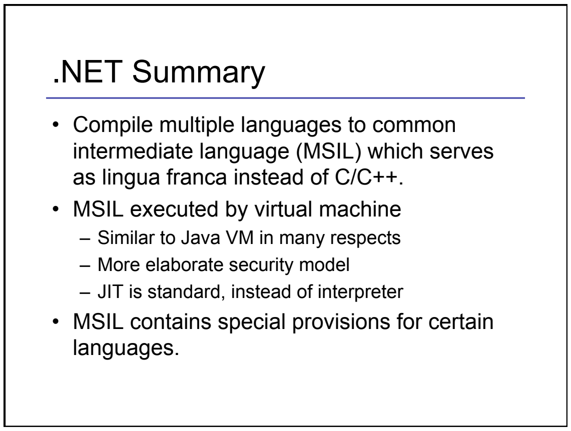
COBOL

```
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".
```



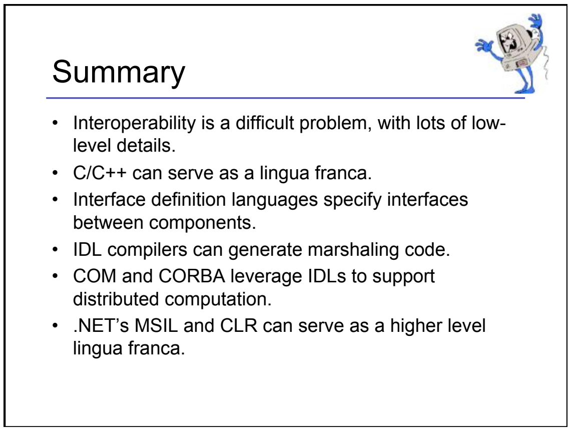
.NET Interoperability

- As examples illustrate, language implementers make CLR Framework Class Hierarchy available within language.
- Compilers can record meta data along with MSIL code.
- Other languages can read meta data to use compiled code from other languages.
- Requires cooperation between compiler writers.



.NET Summary

- Compile multiple languages to common intermediate language (MSIL) which serves as lingua franca instead of C/C++.
- 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

- Interoperability is a difficult problem, with lots of low-level details.
- C/C++ can serve as a lingua franca.
- Interface definition languages specify interfaces between components.
- IDL compilers can generate marshaling code.
- COM and CORBA leverage IDLs to support distributed computation.
- .NET’s MSIL and CLR can serve as a higher level lingua franca.