Comparing DCOM, CORBA and RMI

CORBA

- Same purpose as Java RMI
- Language independent
- Must specify interface in IDL
  - interface definition language
  - mappings defined between IDL and each language
  - If no mapping exists, can’t use that structure

Using an IDL

- Some types are straightforward
  - e.g., int
  - Sizes of ints defined
- Others are more complicated
  - Hash-table
  - Union types

Methods in IDL

- Each parameter can be
  - in
  - out
  - in out

IDL Features

- modules (e.g., packages/ namespaces)
- interfaces
- methods
- attributes
- inheritance (i.e., subtyping?)
- arrays
- sequence
- struct, enum, union, typedef
- consts
- exceptions

In Java

```java
package SimpleStocks;

public interface StockMarket extends java.rmi.Remote {
    double get_price( String symbol ) throws java.rmi.RemoteException;
}
```
In CORBA IDL

module SimpleStocks {
  interface StockMarket {
    double get_price( in string symbol );
  };
};

DCOM IDL

[ uuid(7371a240-2e51-11d0-b4c1-444553540000),
  version(1.0) ]
library SimpleStocks {
  importlib("stdole32.tlb");
  [uuid(BC4C0AB3-5A45-11d2-99C5-00A02414C655),dual]
  interface IStockMarket : Idispatch {
    HRESULT get_price([in] BSTR p1, [out, retval]
      double * rtn);
  };
};

DCOM IDL

[uuid(BC4C0AB3-5A45-11d2-99C5-00A02414C655),]
coclass StockMarket {
  interface IStockMarket;
};

Notes

• Java and CORBA allow exceptions
• DCOM does not
  – all function return HRESUL TS
    * numeric error codes
• All three allow dynamic introspection and invocation

RMI/CORBA interaction

• You can generate RMI stubs that use the same protocol as CORBA (IIOP)
• You can generate IDL from Java classes