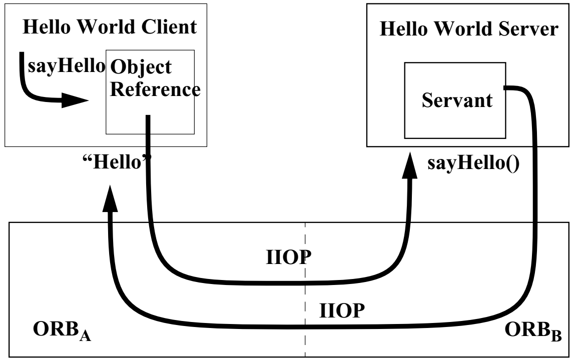
3.6 Corba und Java

- Internet InterORB Protocol (IIOP)



COS Common Object Services (z. B. Naming Service)

ORB Object Request Broker

- **Programmentwicklung**
  1. Beschreibung der Schnittstelle in Corba-spezifischer Schnittstellensprache (IDL)
  2. Implementierung des Dienstnehmers
  3. Implementierung des Dienstleisters
  4. Mittels Stellvertreter-Generator die beiden Stellvertreter erzeugen. Erzeugt werden die Dateien:

     _HelloImplBase.java
     Gerüst für die Methoden des Dienstleisters

     _HelloStub.java
     Stellvetrtreter des Dienstleisters

     Hello.java
     Java-Version der Schnittstellenbeschreibung

     HelloHelper.java
     Hilfsroutinen für die Datenkonversion

     HelloHolder.java
     Hilfsoperationen zur Behandlung von out- und inout-Parametertypen, die in Corba existiern, aber nicht einfach in Java übertragbar sind.

- **Schnittstellenbeschreibung**

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


- **Implementierung des Dienstnehmers**

```
import HelloApp.*;
import org.omg.CosNaming.*;  // NamingContext, NamingContextHelper,
                             // NameComponent
import org.omg.CORBA.*;      // ORB, Object

public class HelloClient {

  public static void main(String args[]) {
    try {
      // create and initialize the ORB for a
      // standalone application
      ORB orb = ORB.init(args, null);

      // get the root naming context
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

```
      // resolve the Object Reference in Naming
      NameComponent nc = new NameComponent("Hello", "");
      NameComponent path[] = {nc};
      Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

      // call the Hello server object and print results
      String hello = helloRef.sayHello();
      System.out.println(hello);

    } catch (Exception e) {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }
  }
}
```


- Implementierung des Dienstleisters

```
import HelloApp.*;
import org.omg.CosNaming.*;  // NamingContext, NamingContextHelper,
                             // NameComponent
import org.omg.CORBA.*;      // ORB, Object

class HelloServant extends _HelloImplBase {
  public String sayHello() {
    return "\nHello world !!\n";
  }
}
```



```
public class HelloServer {

  public static void main(String args[]) {
    // create and initialize the ORB
    ORB orb = ORB.init(args, null);

    try {
      // create servant and register it with the ORB
      HelloServant helloRef = new HelloServant();
      orb.connect(helloRef);

      // get the root naming context
      org.omg.CORBA.Object objRef
           = orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);

      // bind the Object Reference in Naming
      NameComponent nc = new NameComponent("Hello", "");
      NameComponent path[] = {nc};
      ncRef.rebind(path, helloRef);
```



```
      // wait for invocations from clients
      java.lang.Object sync = new java.lang.Object();
      synchronized (sync) {
        sync.wait();
      }

    } catch (Exception e) {
      System.err.println("ERROR: " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

3.7 RPC in Microsoft Windows 2000

❑ Programmentwicklung

1. Beschreibung der Schnittstelle in Microsoft-spezifischer Schnittstellensprache
   - application configuration file (ACF): Enthält Konfigurationsangaben
   - interface-defining file (IDL): Typdefinitionen und Funktionsprototypen (Methodensignaturen)
2. Implementierung des Dienstnehmers
3. Implementierung des Dienstleisters
4. Mittels Stellvertreter-Generator midl das gemeinsame ‘header file’ und die beiden Stellvertreter erzeugen.

◆ ACF

```
[implicit_handle(handle_t Hello_IfHandle)]
interface Hello
{

}
```

◆ IDL

```
[ uuid (C2557720-CA46-1067-B31C-00DD010662DA),
  version(1.0),
  pointer_default(unique)
]
interface Hello {
   const short STRSIZE = 500;
   void hello([in, out, string, size_is(STRSIZE)]
                                    unsigned char * pszInOut);
}
```


◆ Dienstleister

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "Hello.h"     // header file generated by MIDL compiler

#define TRUE  1
#define FALSE 0

void hello(unsigned char *str) {
   printf("hello is called\n");
   sprintf(str, "Hello World!");
   printf("hello returns %s\n", str);
   return;
}  // end function hello
```



```
void _CRTAPI1 main(int argc, char * argv[]) {
    RPC_STATUS status;
    unsigned char * pszProtocolSequence = "ncacn_np";
    unsigned char * pszSecurity         = NULL;
    unsigned char * pszEndpoint         = "\\pipe\\hello";
    unsigned int    cMinCalls           = 1;
    unsigned int    cMaxCalls           = 20;
    unsigned int    fDontWait           = FALSE;

    status = RpcServerUseProtseqEp
                (pszProtocolSequence,
                 cMaxCalls,
                 pszEndpoint,
                 pszSecurity);  // Security descriptor
    printf("RpcServerUseProtseqEp returned 0x%x\n", status);
    if (status) { exit(status); }
```

```
    status = RpcServerRegisterIf
                (Hello_ServerIfHandle, // interface to register
                 NULL,    // MgrTypeUuid
                 NULL);  // MgrEpv; null means use default
    printf("RpcServerRegisterIf returned 0x%x\n", status);
    if (status) { exit(status); }
    printf("The Hello server is in.\n");

    printf("Calling RpcServerListen\n");
    status = RpcServerListen(cMinCalls,
                             cMaxCalls,
                             fDontWait);
    printf("RpcServerListen returned: 0x%x\n", status);
    if (status) { exit(status); }
```



```
    if (fDontWait) {
       printf("Calling RpcMgmtWaitServerListen\n");
       status = RpcMgmtWaitServerListen();  // wait operation
        printf("RpcMgmtWaitServerListen returned: 0x%x\n", status);
       if (status) { exit(status); }
    }

}  // end main()
```



```
/*********************************************************************/
/*                  MIDL allocate and free                           */
/*********************************************************************/

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len)
{
    return(malloc(len));
}

void __RPC_USER midl_user_free(void __RPC_FAR * ptr)
{
    free(ptr);
}
```


- Dienstnehmer

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "Hello.h"    // header file generated by MIDL compiler

void _CRTAPI1 main(int argc, char **argv) {
   RPC_STATUS status;            // returned by RPC API function
   unsigned char helloIn[STRSIZE];
```

```
unsigned char * pszUuid               = NULL;
unsigned char * pszProtocolSequence = "ncacn_np";
   // connection through named pipe
unsigned char * pszNetworkAddress   = NULL;
   // local host
 unsigned char * pszEndpoint          = "\\pipe\\hello";
   // name of endpoint
 unsigned char * pszOptions           = NULL;
 unsigned char * pszStringBinding     = NULL;
   // return of binding handle
```



```
/* Use a convenience function to concatenate the elements of  */
/* the string binding into the proper sequence.                */
status = RpcStringBindingCompose(pszUuid,
                                 pszProtocolSequence,
                                 pszNetworkAddress,
                                 pszEndpoint,
                                 pszOptions,
                                 &pszStringBinding);
if (status) {
    printf("RpcStringBindingCompose returned 0x%x\n", status);
    printf("pszStringBinding = %s\n", pszStringBinding);
    exit(status);
}
```



```
/* Set the binding handle that will be used to bind to the server. */
    status = RpcBindingFromStringBinding(pszStringBinding,
                                         &Hello_IfHandle);
    if (status) {
        printf("RpcBindingFromStringBinding returned 0x%x\n",
               status);
        exit(status);
    }

    /* RPC is now initialized.                               */
```



```
/* Call remote procedures as if                            */
/* they were local procedures.                             */

RpcTryExcept {
    hello(&helloIn[0]);
    printf("\n%s\n", helloIn);    // no, continue
}
RpcExcept(1) {
    printf("Runtime reported exception %ld\n",
           RpcExceptionCode() );
}
RpcEndExcept
```

```
    /*  The calls to the remote procedure are complete.  */
    /*  Free the binding handle.                         */
    status = RpcBindingFree(&Hello_IfHandle);
       // remote calls done; unbind
    if (status) {
       printf("RpcBindingFree returned 0x%x\n", status);
       exit(status);
    }

    exit(0);

}  // end main()
```



```
/****************************************************************/
/*                 MIDL allocate and free                       */
/****************************************************************/

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len)
{
    return(malloc(len));
}

void __RPC_USER midl_user_free(void __RPC_FAR * ptr)
{
    free(ptr);
}
```