CORBA Common Object RequestBroker Architecture

Distribuované systémy 2006

Úvod

- **CORBA - Common Object Request Broker Architecture**
- standard pro vytváření požadavků na objektové volání metod v síti
- OMG – Object Management Group – vývoj specifikace CORBA
- vytvářena s cílem podporovat různé sítě, operační systémy a jazyky
- na rozdíl od RMI není limitována programovacím jazykem (Java)
- CORBA není jazyk, ale specifikace jak mají objekty vzájemně působit

Historie

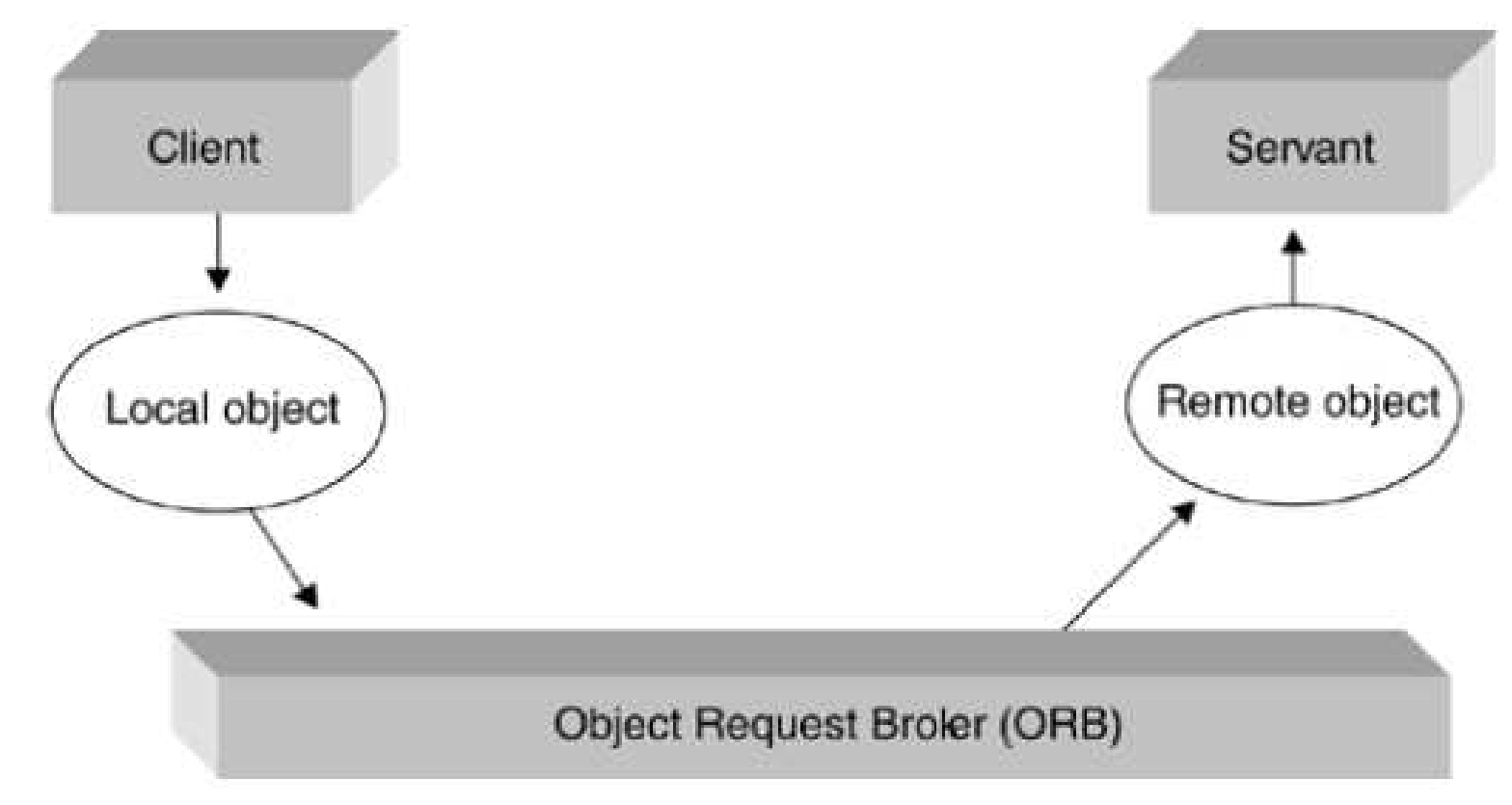
1990 OGM vydalo Object Management Architecture Guide (OMA Guide), obsahující základní specifikace, ale bylo to ještě nedostatečné.

1991 Vzniká Common Object Request Broker Architecture (CORBA) – verze 1.0, zmiňuje bezpečnost, přibyly Services (služby) a Facilities (vybavení).

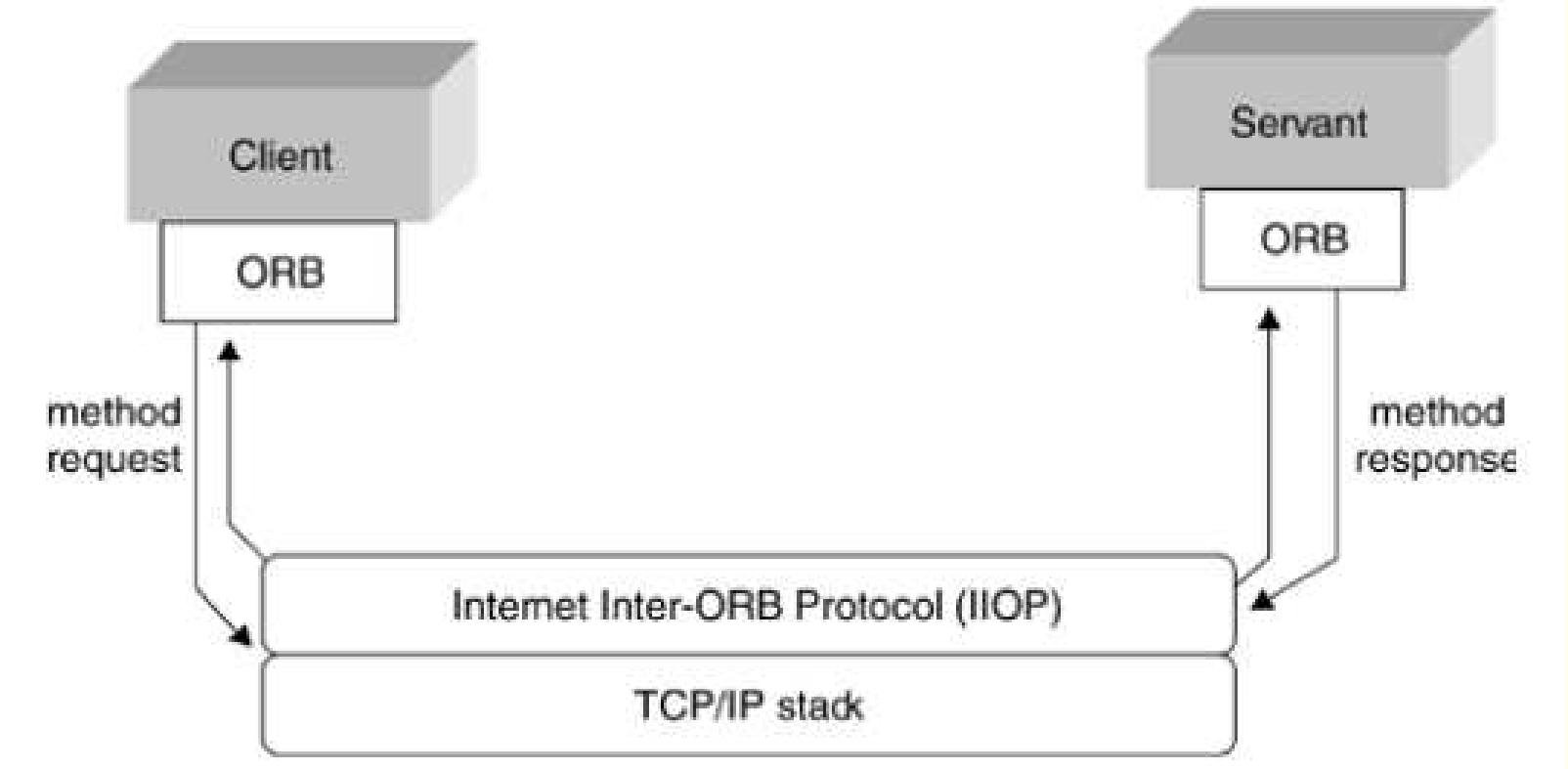
1994 CORBA verze 2.0 – Implementation repository (uložení implementací objektů), rozšíření Services a Facilities.

2000 CORBA 3 - obsahuje Internet Integration (Firewall Specification, Interoperable Name Service), Quality of Service Controls (Minimum, Fault-Tolerant, and Real-Time CORBA), a CORBA components and CORBA scripting (CORBA components bude něco jako Enterprise JavaBeans, akorát ve více jazycích)

Úvod


Client
Servant
Local object
Remote object
Object Request Broler (ORB)

Úvod


Client
ORB
Servant
ORB
method
request
method
response
Internet Inter-ORB Protocol (IIOP)
TCP/IP stack

IDL

- IDL – Interface Definition Language – vyjádření schématu pro popis služeb. jazyk podporující CORBA musí umět implementovat IDL schéma
- IDL se použije pro automatické generování modulů daného jazyka, které umožní volat objekty.
- IDL je podporováno následujícími jazyky
  - C, C++, Smalltalk, COBOL, Ada, Java

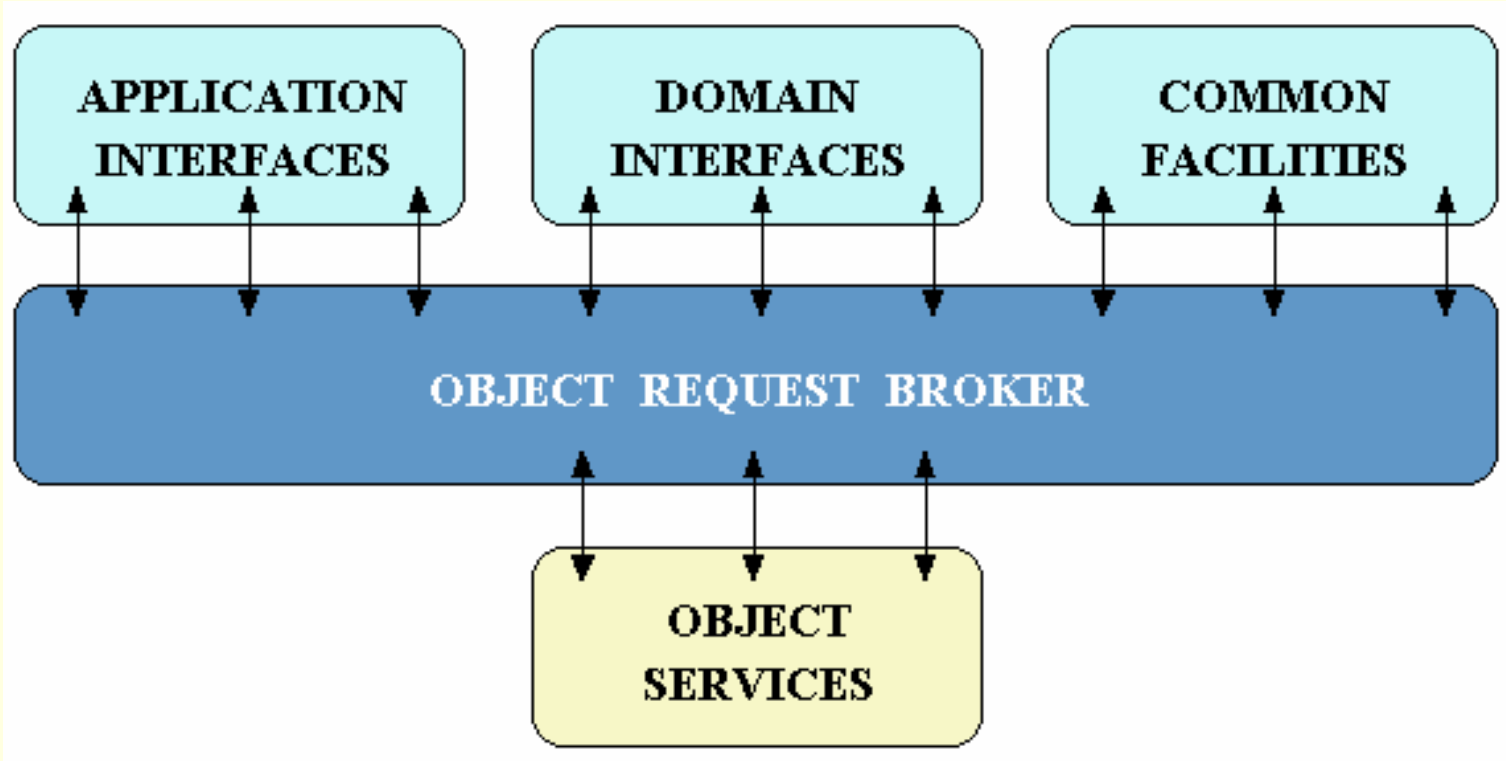
ORB Object request Broker

- ORB – Object Request Broker – realizuje rozhraní mezi vzdáleným objektem a programovým klientem
- ORB zpracovává požadavky na objektové služby a přenáší odezvy.
- servant – implementace služby

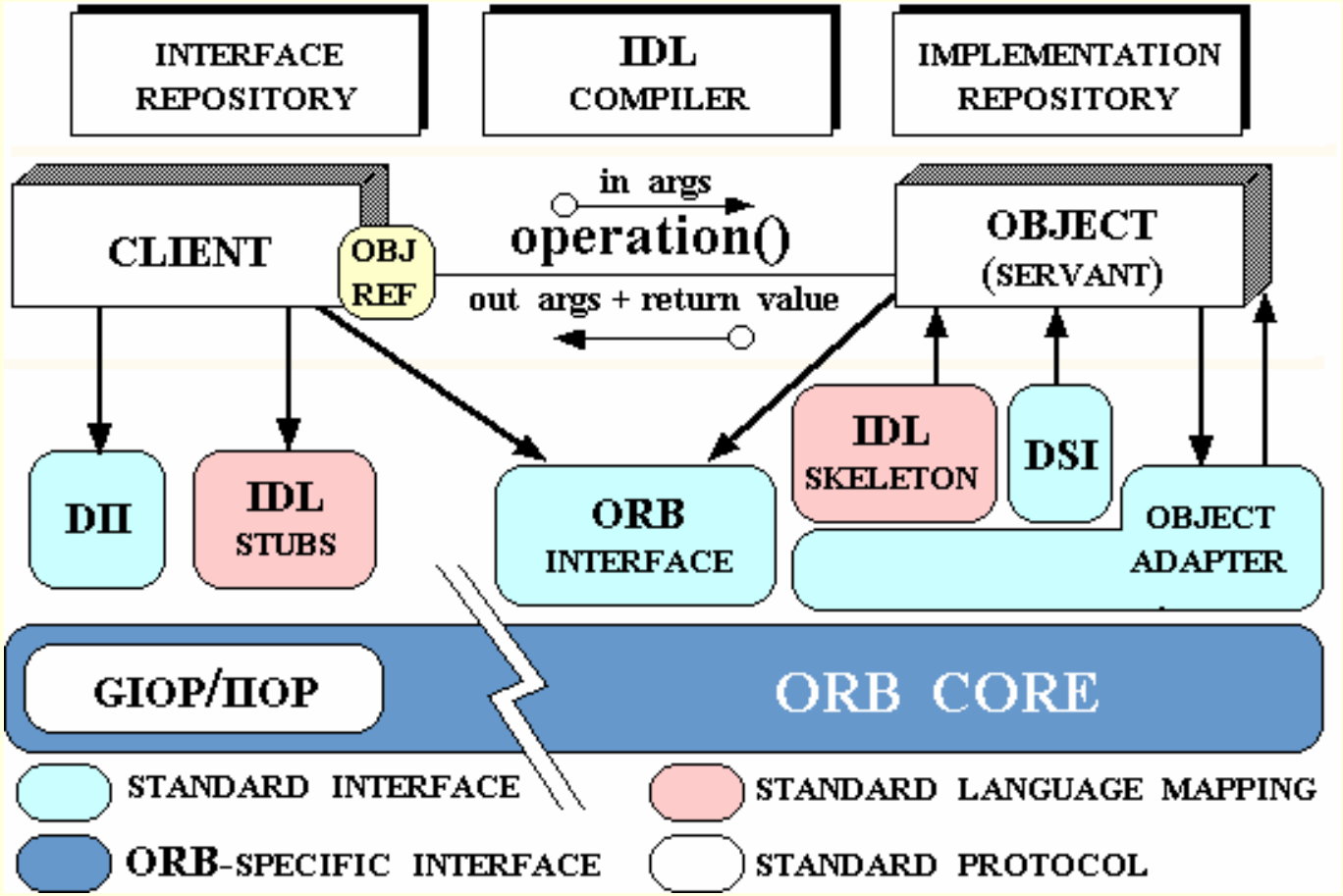
IIOP Internet Inter-ORB Protocol

- Objekty jsou chápány tak, že metody jsou volány jako by to byly normální lokální metody
- ORB zajišťuje komunikaci mezi klientem a servantem pomocí IIOP – Internet Inter-ORB Protocol.

Referenční architektura



Referenční architektura



CORBA architektura

- Objekt – programová entita, obsahující identitu, interface a implementaci ynámou jako servant
- Servant – implementace entity, která definuje operace pro podporu CORBA IDL rozhraní
- Object Request Broker (ORB) – mechanizmus pro transparentní komunikaci mezi klientem a implementací cílového objektu. Skrývá implementační detaily volání metod před klientem. ORB je schopen nalézt implementaci objektu, transparentně jej aktivovat, doručit požadavek i odpověď.

CORBA architektura

- ORB interface – ORB je logická entita, která může být implementována různým způsobem (jedním nebo více procesy, souborem knihoven). CORBA definuje abstraktní interface pro ORB. tento interface provádí různé pomocné funkce, umožňující transparentní volání objektů.
- CORBA IDL spojky a skeletony – slouží k propojení aplikací serveru a klienta s ORB. Transformace mezi CORBA IDL definicemi a cílovým programovacím jazykem je automatizována CORBA IDL překladačem.

CORBA architektura

- Dynamic Invocation Interface (DII) – rozhraní dovoluje klientovi přímo přistupovat k rozhraní ORB. DII používají aplikace, které volají objekty dynamicky bez požadavku využití specifických spojek IDL rozhraní. To dovoluje provádět nejen standardní blokované operace, ale i neblokované synchronní (oddělení send a receive) a onewaz (send-only) volání.
- Dynamic skeleton Interface (DSI) – je to analogie DII na straně serveru. Dovoluje ORB doručit požadavky na implementaci objektu, kdy se při překladu neví, jakého typu objekt je. Klient vytvářející požadavek neví, používá-li implementace IDL skeleton nebo dynamický skeleton.

CORBA architektura

- Object Adapter – asistuje ORBu při doručování požadavků na objekt a s aktivací objektu. Spojuje implementaci objektu s ORBem.

CORBA základ komunikace

- Schopný zpracovat základní komunikaci mezi klientem a objektem
- Základní komunikace spočívá v tom, že je schopen poslat výzvu serveru a odpověď zpět klientovi
- ORB nabízí pouze několik služeb, jednou ze služeb je manipulace s referencí na objekt
- ORB dále nabízí operace marshall a unmarshal reference objektu
- ORB nabízí hledání služeb dostupných objektu (získání počátečního odkazu na implementaci objektu)
- Rozhraní mezi ORB a klientem je popsáno IDL – kód potřebný pro zpracování komunikace mezi klientem, serverem a ORB je generováno IDL překladačem

Dynamic Invocation Interface

- Používá se v případech, kdy nejsou dostupná klientovi staticky definovaná rozhraní
- Za běhu se musí zjistit, jak vypadá rozhraní k objektu a pak se vytvoří požadavek na volání objektu
- Základní operace je INVOKE, která podle reference na objekt, identifikátoru metody, seznamu vstupních hodnot vytvoří seznam výstupních proměnných vytvořených voláním

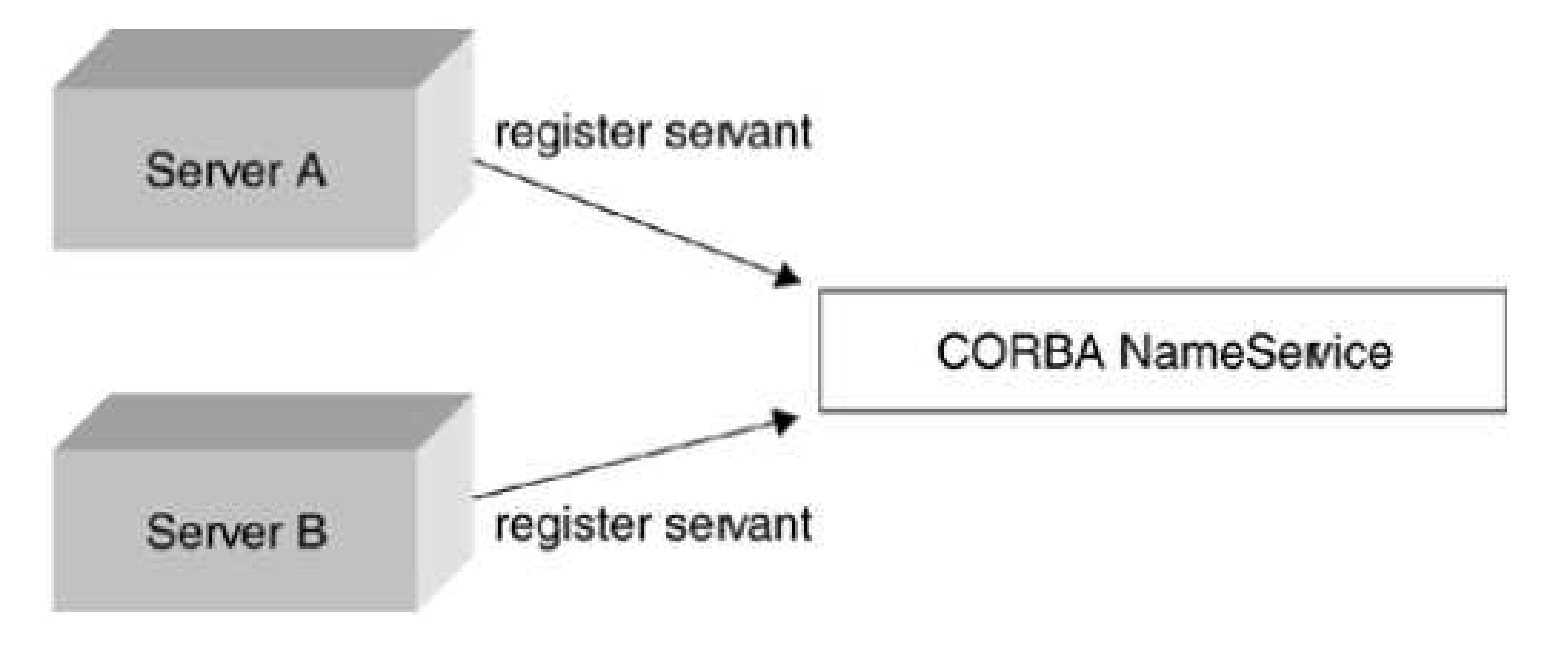
Implementation respository

- uložení všech definic volání – umožňuje volat metody dynamicky
- IDL překladač přiřazuje při překladu respozitory identifier – jednoznačný identifikátor
- nemusí být unikátní, odvozen od jména rozhraní a metod
- obsahuje vše potřebné k realizaci a aktivaci objektů
- závislé na ORB a OS – obtížné zajistit standardní implementaci

Služby

- softwareové služby jsou popisovány schématem a realizovány servantem
- servant je program, který se registruje u „lookup“ služeb, takže ostatní programy pod CORBA mohou získat přístup k jeho službám.
- Typicky CORBA server vytváří CORBA servant
  - pak je schopný vytvořit ORB pro servant a registrovat služby pro přístup ostatních klientů.

Služby


Server A
register servant
CORBA NameService
Server B
register servant

Typy služeb

- **collection service** – prostředky pro seskupování objektů do seznamů (fronty, zásobníky, množiny)
- **query service –** vytváření kolekcí objektů, které mohou být řazeny do front, seznamů, dotaz může vrátit odkaz na objekt nebo množinu objektů
- **concurency control service –** nabízí prostředky pro konkurentní přístup ke sdíleným objektům
- **transaction service –** dovoluje vytvářet transakce**,** volání více metod v transakci, plošné transakce, vnořené transakce

Typy služeb

- **event service –** podporuje asynchronní komunikaci, umožňuje přerušit činnost klientů i serverů při výskytu specifické události
- **notification service –** prostředky pro asynchronní komunikaci
- **externalizace –** marshalling objektů tak, že mohou být uloženy na disk nebo posílány sítí (serializace objektů v Javě) – převod objektu na posloupnost slabik
- **life cycle service –** služby spojené s vytvořením, rušením, kopírováním a přesunem objektů
- **licencing service –** zajištění specifické licenční politiky, licence vyjadřuje práva klienta vzhledem k objektu

Typy služeb

- **naming service –** označování objektů jmény, mapování na ID objektu
- **property service –** základní prostředky pro popis objektů, popisováno párem (atribut, hodnota), popisuje objekt, není stavem objektu
- **trading service –** dovoluje objektu nabízet co umí (prostřednictvím definice rozhraní), dovoluje klientům vyhledávat služby
- **persistence service –** prostředky pro uložení informace na disk ve formě pamě´tových objektů

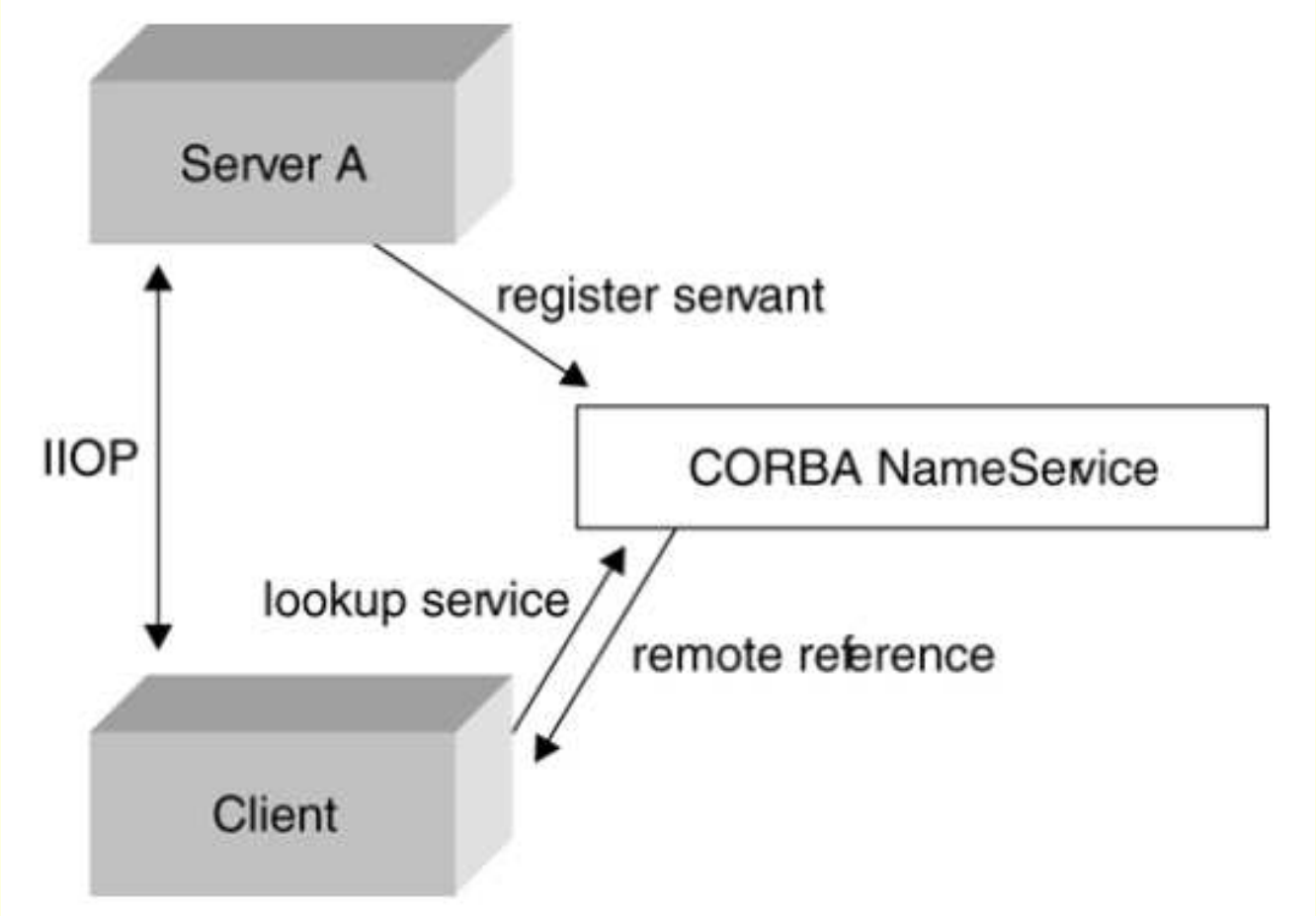
Typy služeb

- **relation service –** prostředky pro vytvoření vztahu mezi dvěma a více objekty, podpora pro organizaci objektů
- **security service –** autentikace, autorizace, audit, bezpečná komunikace, administrace, nepopiratelnost
- **time service –** běžný čas se specifikovaným intervalem nepřesnosti
- Všechny služby jsou specifikovány pomocí IDL, dovoluje oddělit specifikaci a implementaci

Klienti

- klienti využívají lookup služeb pro vyhledání služeb servantů
- služby se mohou v systému přesunovat, transparentnost přesunu zajišťují lookup služby

Jmenné služby



CORBA komunikace

- původně synchronní, postupně předělána na asynchronní
- Modely volání objektu
- **synchronní** – klient je blokován dokud nepřijme odpověď
  - sémantika *at-most-once*
  - vhodná pokud volající čeká odpověď, dostane-li odpověď, bylo volání úspěšné
  - nečeká-li volající na výsledek, je vhodné co nejdříve pokračovat (obdoba asynchronního RPC)
- **one way request** – metoda volání v CORBA, není zaručeno vyvolání objektu (*best effort*)
- **deffered synchronous request** – po odeslání požadavku klient pokračuje
  - na odpověď čeká později
  - sémantika *at-most-once*

CORBA události

- kromě komunikačních služeb je třeba mít k dispozici mechanizmus, který by oznamoval události (výskyt událostí)
- event service – každá událost je spojena s datovou položkou (odkaz na objekt, hodnota specifická pro aplikaci)
- událost je produkována modulem **supplier**, přijímána modulem **consumer**
- události jsou doručovány **kanálem událostí**

CORBA události

- je používán **push model nebo poll model**
  - push – consumer čeká pasivně na událost
  - pool – dotazování na výskyt události
- **nevýhody** – consumer i suppier musí být připojení ke kanálu událostí – události se mohou ztratit
- **problém filtrování** – kanál je společný, consumer musí filtrovat data nebo se musí vytvořit pro každý typ události oddělený kanál

Notification service

- schopnost filtrování událostí
- prevence propagace událostí, o které nemají příjemci zájem
- propagace událostí je nespolehlivá

Přenos zpráv

- **transient** (přechodná) komunikace – zpráva je v komunikačním kanálu pouze tehdy, pokud běží klient i server
- **persistentní** (přetrvávající) komunikace – zpráva je v komunikačním kanálu dokud nemůže být doručena (*message queueing model*)

Přenos zpráv

- podporuje message queueing model
  - callback model
  - pooling model
- callback model – klient zpracovává objekt, který implementuje interface obsahující callback metody
  - metody jsou volány z nižší vrstvy při asynchronní komunikaci
  - nemají vliv na původní implementaci objektu
  - server realizuje synchronní požadavky

Přenos zpráv

- konstrukce asynchronního volání
  - náhrada původního rozhraní novým, realizovaným na straně klienta
  - specifikace metod, které může klient volat
  - callback interface
  - rozdělení metody na část vstupní a výstupní (dvě metody)
  - send(in) – voláno klientem
  - reply(out) – voláno ORBem

Přenos zpráv

- pooling model – transformace původní synchronní metody na asynchronní
  - rozdělení původního rozhraní na
  - send_pool(in) – voláno klientem
  - reply_pool(out) – voláno kolientem

Interoperability

- problémy se spoluprací ORB od různých výrobců řeší zavedení inter-ORB protokolu
- General Inter-ORB Protocol – GIOP
- předpokládá, že vlastní přenosy zajistí existující protokol (TCP) – spolehlivý, spojově orientovaný, tok slabik
- realizace GIOP nad TCP se označuje Internet Inter-ORB Protocol (IIOP)

General Inter ORB protocol GIOP

- GIOP definuje 8 typů zpráv
  - **Request** – úplná požadavek (odkaz na objekt, jméno metody, vstupní parametry, identifikátor (párování s reply)
  - **Reply** – identifikátor, návratové hodnoty, výstupní parametry
  - **LocateRequest** – dotaz na umístění objektu – objekty jsou v implementation respository
  - **LocateReply** – obsahuje server kde je objekt
  - **CancelRequest** – rušení Request nebo LocateRequest
  - **CloseConnection** – požadavek na ukončení spojení
  - **MessageError** – obsahuje záhlaví zprávy, která způsobila chybu (ICMP)
  - **Fragment** – část dlouhé zprávy – identifikuje originální zprávu obsahuje údaje o fragmentaci

Procesy

- server
- klient
- proxy – propojení aplikace klienta s ORB – DII (Dynamic Invocation Interface)
- Interceptor – změna volání
  - request level interceptor
  - message level interceptor
  - interceptory jsou viděny pouze ORBem

Interface Definition Language IDL

- primitivní datové typy –
  - (void (Void),
  - boolean (Boolean),
  - wchar (Char),
  - octet (Byte),
  - short (Short), long (Int),
  - long long (Long),
  - float (Float),
  - double (Double),
  - string (String),
  - wstring (String))
- konstruované datové typy – (pole, sekvence, datové struktury)

Moduly

- slouží ke sdružování logicky vázaných rozhraní (obdoba package)

```
module fce
{
     interface fce_XX
    {
    };
    interface fce_YY
    {
    }
}
```

Rozhraní

- popisuje vzdálený objekt, nabízené metody, členské proměnné, konstanty

```
interface fce_XX
{
   void set_XX( in float x );
   float get_XX( );
};
```

Atributy

- rozhraní mohou mít definovány proměnné – členské proměnné
- mohou být modifikovatelné nebo nemodifikovatelné (mutable, immutable)

```
interface fce_XX
{
   attribute short result;
   readonly attribute short id;
}
```

Operace

- operace jsou funkce
  - in – neměnný, vstupní
  - out – modifikovatelný, výstupní
  - inout – kombinace předchozích

Zpracování výjimek

- při výskytu chyby se hlásí výjimka – volání třídy (ne jen příznak chyby a její typ)

```
module X
{
    exception A
    {
    };
    exception B
    {
    };
    interface R
    {
            void M(in short p, out short q) raises
            (A, B);
    };
}
```

Příklad

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

Příklad - překlad

- idlj –f server Hello.idl
- idlj –f client Hello.idl
- idlj –f all Hello.idl

HelloOperations.java – operace, které musí COBRA servant realizovat
_HelloStub.java – musí být děděn v modulu, který implementuje metody dle HelloOperations
Hello.java
HelloHelper.java
HelloHolder.java
HelloPOA.java

HelloException.java
HelloExceptionHelper.java
HelloExceptionHolder.java

HelloOperations.java

```
package HelloApp;
/**
* HelloApp/HelloOperations.java
* Generated by the IDL-to-Java compiler (portable),
*  version "3.1" from Hello.idl
*/

public interface HelloOperations
{
  String sayHello ();
  void shutdown ();
} // interface HelloOperations
```

Hello.java

```
package HelloApp;
/**
* HelloApp/Hello.java
* Generated by the IDL-to-Java compiler (portable),
* version "3.1" from Hello.idl
*/
public interface Hello extends HelloOperations,
  org.omg.CORBA.Object,
  org.omg.CORBA.portable.IDLEntity
{
} // interface Hello
```

HelloServant.java

```
  import HelloApp.*;
import org.omg.CosNaming.*;
import java.util.Properties; …
class HelloServant extends HelloPOA {
  private ORB orb;
  public void setORB(ORB orb_val) {
    orb = orb_val; }
   // implements sayHello() method
   public String sayHello() {
     return "\nHello world !!\n";
  }
  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
} //end class
```

HelloServer.java

```
public class HelloServer {
  public static void main(String args[]) {
   try{ // create and initialize the ORB
   ORB orb = ORB.init(args, null);
   // get reference to rootpoa & activate the POAManager
   POA rootpoa =
  (POA)orb.resolve_initial_references("RootPOA");
   rootpoa.the_POAManager().activate();
   // create servant and register it with the ORB
   HelloServant helloServant = new HelloServant();
   helloServant.setORB(orb);
   // get object reference from the servant
   org.omg.CORBA.Object ref =
      rootpoa.servant_to_reference(helloServant);
   // and cast the reference to a CORBA reference
   Hello href = HelloHelper.narrow(ref);
```

HelloServer.java

```
// get the root naming context
// NameService invokes the permanent name service
org.omg.CORBA.Object objRef =
      orb.resolve_initial_references("NameService");
// Use NamingContextExt, which is part of the
// Interoperable Naming Service (INS) specification.
NamingContextExt ncRef =
      NamingContextExtHelper.narrow(objRef);
// bind the Object Reference in Naming
String name = "Hello";
NameComponent path[] = ncRef.to_name( name );
ncRef.rebind(path, href);
System.out.println("HelloServer is ready ...");
// wait for invocations from clients
orb.run();
```

HelloClient.java

```
public class HelloClient{
  static Hello hello;
  public static void main (String args[]){
     try{ ORB orb = ORB.init(args, null);
      org.omg.CORBA.Object objRef =
           orb.resolve_initial_references
                                  ("NameService");
     NamingContextExt ncRef =
           NamingContextExtHelper.narrow(objRef);
     hello = HelloHelper.narrow(ncRef.resolve_str
                ("Hello"));
     System.out.println(hello.sayHello());
     hello.shutdown();
```

Program servant HelloServant

Vytvoření ORB

```
ORB orb = ORB.init(args, null);
```

Vytvoření servantu

```
HelloServant servant = new HelloServant();
```

Připojení k ORBu

```
orb.connect(servant);
```

Vytvoření odkazu na službu (Name service)

```
org.omg.CORBA.Object object =
    orb.resolve_initial_references("NameService“)
```

Program servant - HelloServant

Zúžení obecné služby na namingContext

```
NamingContext namingContext = NamingContextHelper.narrow(obje
```

Vytvoření pojmenované komponenty pro náš servant

```
NameComponent component = new NameComponent („Hello", "");
```

Vytvoření prostoru pro pole deskriptorů kontextu

```
NameComponent componentList[] = { component } ;
```

Seznámení jmenných služeb s naším novým rozhraním

```
namingContext.rebind(componentList, servant);
```

Program klient HelloClient

Vytvoření komponenty pro servant

NameComponent component = new NameComponent („Hello", "");

Vytvoření prostoru pro pole deskriptorů kontextu

NameComponent componentList[] = { component } ;

Vyhledání komponenty podle jména

org.omg.CORBA.Object remoteRef
= namingContext.resolve(componentList);

Vyhledání konkrétního odkazu na instanci servantu.
Modul Helper mapuje elementy IDL na Java kód.

Hello s = HelloHelper.narrow (remoteRef);

Zpracování příkladu

idlj –fall Hello.idl

javac *.java

orbd -ORBInitialPort 1050 -ORBInitialHost localhost

tnameserv -ORBinitialPort 1050

java HelloServer -ORBInitialHost localhost -ORBInitialPort 1050

java HelloClient -ORBInitialHost localhost -ORBInitialPort 1050