CORBA

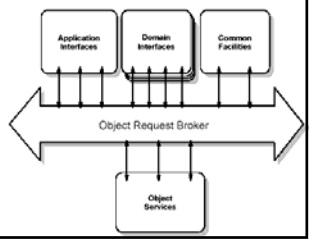
Common Object Request Broker Architecture

OMA – Object Management Architecture

- Standard OMG (Object Management Group) pentru aplicaţii în medii distribuite
- CORBA este elementul cheie al OMA
- OMA este compus din
  - Model de Obiect (**Object Model**) pentru un mediu eterogen
  - Model de Referinţă (**Reference Model**) pentru interacţiunile dintre obiecte.

Componente OMA

•Object Services – interfete generale folosite de multe aplicatii
- Naming Service
- Trading Service
- Life Cycle Service
- Event Service
- Time Service
- Security Service
- Concurrency Control Service



OMA (2)

Componente OMA (2)

- Common Facilities – interfete cu functionalitati orientate spre aplicatii
  - Ex. Distributed Document Component Facility (DDCF) pentru prezentare si interschimb de documente
- Domain Interfaces – specifice domeniului aplicatiei
  - Ex. Product Data Management (PDM) Enablers pentru domeniul fabricatiei
- Application Interfaces – specifica fiecarei aplicatii
- ORB – comunicarea intre clienti si obiecte server; ascunde:
  - localizarea obiectului
    - in acelasi proces
    - in alt proces pe acceasi masina
    - pe alta masina
  - implementarea obiectului
    - limbajul folosit pentru scrierea lui (C, C++, Java, Ada95, COBOL,Smalltalk)
    - sistemul de operare (Win32,UNIX,MVS, VxWorks, Chorus, LynxOS)
    - caracterisiticile masinii
  - starea executiei
    - activat sau neactivat (porneste automat in al doilea caz)
  - mecanismele de comunicare cu obiectul
    - apel local, memorie partajata, TCP/IP, IPX/SPX, FDDI, ATM, Ethernet, Fast Ethernet etc.

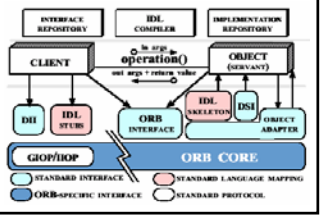
ORB

Clientul executa taskuri specifice aplicatiei obtinand referinte de obiecte si invocand operatiile lor:
- obiectele pot fi situate in aceeasi masina cu clientul sau la distanta
- accesul transparent, similar unuia local - ***object.operation(args)***

Obiectul implementeaza un serviciu descris de o interfata IDL
- identificat printr-o **referinta de obiect** unica
- referinta este "**opacă**" pentru client
- Format standardizat; Ex. IIOP

Obtinere referinte:
- La creare obiect – rezultat al obiect **fabrică** (factory).
- Prin directory service
- Prin conversia la/de la un sir in memoria permanenta.

Referintă initială:
- apel **resolve-initial-reference** cu parametrul **Name-Service**
- se obtine referinta unui **serviciul de nume**



Servant implementeaza operatiile descrise de IDL, in forma
- unul sau mai multe obiecte (Java, C++)
- tipuri abstracte (C)

ORB interface - interfata la serviciile de obiecte (ciclu de viata, nume etc.)

IDL Stubs si Skeleton – interfete de invocare a serviciilor
- marshalling si un-marshalling pentru parametrii invocarilor

DII - Dynamic Invocation Interface
- permite generarea invocarilor la momentul executiei
- foloseste descrieri dintr-un depozit de interfete (IR - Information Repository) necunoscute la compilare operatia creata poate fi invocata in trei moduri
  - **sincron** (clientul se blocheaza in asteptarea raspunsului)
  - **sincron intarziat** (invoca cererea, continua prelucrarea si colecteaza ulterior raspunsul; utila pentru lansarea mai multor cereri paralele)
  - **oneway** (nu exista raspuns)

DSI - Dynamic Skeleton Interface - echivalentul DII la server
- permite ORB sa livreze cereri unui server care nu cunoaste la compilare interfata implementata

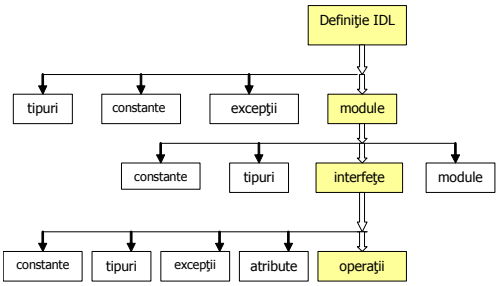
Interface Repository (IR)
- furnizeaza la executie informatii despre interfetele IDL
- IR = obiect CORBA ale carui operatii pot fi invocate simiular altor obiecte

Implementation Repository
- **contin informatii care permit activarea serverelor si proceselor server**
- **specifice sistemuli de operare si implementarii ORB**
- **pastreaza informatii despre alocarea resurselor, securitate, modurile de activare**

Object Adapter (OA)
- **Înregistrarea obiectelor** – OA include operatii care permit unor entităti să se înregistreze ca implementări de obiecte CORBA.
- **Generarea referintelor** de obiecte CORBA
- **Activarea procesului server** – dacă este necesar, OA porneste procesele server în care pot fi activate obiectele
- **Activarea obiectelor** – OA activează obiectele, dacă ele nu sunt deja active la sosirea cererilor.
- **Demultiplexarea cererilor** – OA asigura ca cererile pot fi primite prin conexiuni multiple, fără blocare nesfârsită a vreunei conexiuni
- **Apeluri de obiecte** (object upcalls) - OA distribuie cererile către obiectele înregistrate.

OMG IDL - Interface Definition Language

- limbaj declarativ
- asigura independenta fata de limbaj implementare
- nucleul: declaratii interfete



Tipuri

Tipuri de bază

long (signed, unsigned) – 32 biti
long long (signed, unsigned) – 64 biti
short (signed, unsigned) – 16 biti
float, double, long double
char, wchar (wide char – 16 biti)
boolean
octet
enum
any – poate reprezenta orice tip de bază sau construit de utilizator.

Tipuri construite

```
struct catalog {course curs;
                grade nota;
}

union Dataltem switch (char){
   case 'c': long count;
        case 'a': float amount;
        default: char initial;
}

typedef char message[80];
```

Interfete

```
interface nume[:baza {, baza}]{
   … declaratii de constante, tipuri, exceptii, atribute, operatii, …
};
```

Exemplu 1:

```
interface Factory{
        Object create();
};
```

Exemplu 2 (mostenire):

```
interface Person {
        readonly attribute string name;
};
interface student: Person {
        attribute Professor advisor;
        Enrollments load (in semester when);
};
```

Operatii IDL

- O operatie denotă un **serviciu.** Descrisă prin:
  - identificatorul operatiei
  - tipul rezultatului (orice tip IDL)
  - descrierea fiecărui parametru
    - nume
    - tip
    - (mod) directie
    - **in** client->server
    - **out** server->client
    - **inout** ambele directii
  - **exceptiile** - clauza **raises**

Operatii

```
result_type op_name ( [direction type name {, directions type name} ])
      [raises ([exception {, exception }])]
```

Ex.1

```
Students roster (in Semester when);
   void enroll (in Course course_to_take, out Course prereqs);
   long pop() raises (underflow);
```

Atribute

```
[readonly] attribute <datatype><name>
```

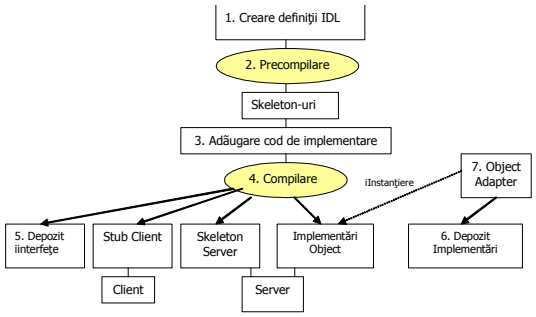
Ex. 2

```
interface Adder {
   …
   readonly attribute long sum; };
```

Corespondența între IDL şi C++

OMG IDL Type	*C++ Mapping Type*
long, short	long, short
float, double	float, double
enum	enum
char	char
boolean	bool
octet	unsigned char
any	Any class
struct	struct
union	class
string	char*
wstring	wchar_t*
sequence	class
fixed	Fixed template class
object reference	pointer or object
interface	class

CORBA static

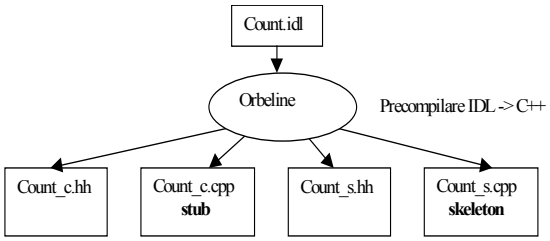


Exemplu: Count

- **Serverul** suportă o singură metodă numită **increment**, care incrementează valoarea unei variabile numită **sum** si întoarce clientului valoarea acesteia.
- Interfata serverului (in fisierul count.idl):

```
module Counter
{ interface Count
   { attribute long sum;
     long increment();
   };
};
```

Rezultate precompilare (VisiBroker pentru C++)



count_s.cpp	skeleton-ul server pentru metodele clasei Count.
count_s.hh	include definitiile de clase pentru skeleton si server.
count_c.cpp	contine o clasă Count = proxy pentru obiectul Count metoda _bind pentru localizare server Count.
count_c.hh	este fisierul antet pentru client.

Implementare server

Include implementare
- **Interfete**
- **Program principal**

Parte din Count_s.hh

```
class _sk_Counter                          // coresp modul Counter
{   public:
    class _sk_Count: public Counter::Count  // coresp interf Count
     { virtual CORBA::long sum()=0;         //citeste atribut
       virtual void sum (CORBA::long val)=0; //scrie atribut
       virtual CORBA::long increment()=0;   //op. Increment
       . . .                                //alte operatii skeleton
     };
};
```

Implementare CountImpl

```
// countimp.h definitia clasei pentru implementarea lui Count
// VisiBroker pentru C++

#include <count_s.hh>
class CountImpl: public _sk_Counter:: _sk_Count
{
   private:
        long _sum;
   public:
        CountImpl (const char * object_name=NULL);
        CORBA::long sum();
        void sum(CORBA::long val);
        CORBA::long increment();
};
```

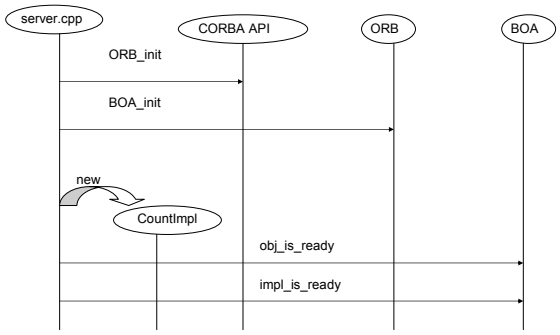
Implementarea **CountImpl**

```
//countimp.cpp Count Implementation, VisiBroker pentru C++
#include "countimp.h"
//Constructor
CountImpl::CountImpl (const char *object_name)
            :_sk_Counter :: _sk_Count(object_name)
{ cout<<"Count object created"<<endl;
  this->_sum=0;
}

//citeste valoarea atribut sum
CORBA::long CountImpl::sum()
{  return this->_sum; }

//scrie valoarea atributului sum
void CountImpl::sum(CORBA::long val)
{  this->sum=val; }

//incrementează suma
CORBA::long CountImpl::increment()
{  this->_sum++;
   return this->_sum;
}
```

Programul principal Server

```
//server.cpp Count server
#include “countimp.h”
int main (int argc, char * const * argv)
{ try
     { // initializare ORB
       CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);
       //init BOA
       CORBA:: BOA_ptr boa = orb->BOA_init(argc, argv);
       // creează obiect Count
      Counter::Count_ptr count= new CountImpl (“MyCount”);
       // exportă noul obiect – înregistrează cu BOA
       boa->obj_is_ready (count);
       // gata să servească cererile
       boa->impl_is_ready();
     }
catch (const CORBA::Exception& e)
     { cerr<<e<<endl; exit (-1)}
return (0);
}
```

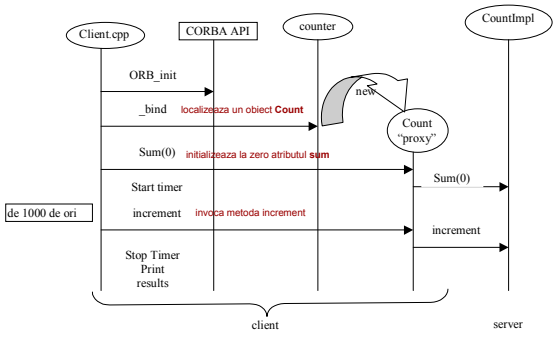
Actiunile programului server



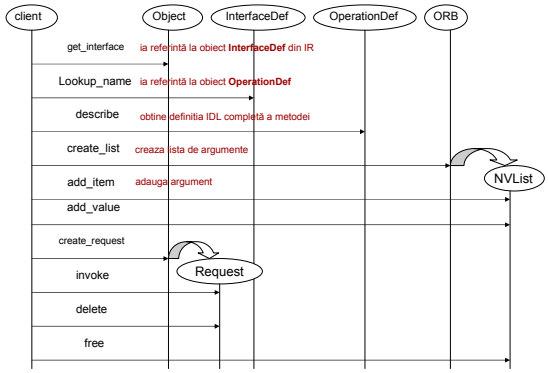
Clientul Count

```
//Client.cpp Count static client, VisiBroker pentru C++
#include <count_c.hh>
#include …
struct timeb timebuff;
double startTime, stopTime;
int main (int argc, char * const* argv)
{ try
      {cout << "initialize ORB" << endl;
      CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);
      cout << "Binding to Count Object"<<endl;
      Counter::Count_var Counter=Counter::Count::_bind(“MyCount”);
      /*Counter va contine un pointer la “proxy” intermediarul pentru serverul CountImpl */
      Counter->sum((long)0);
      ftime(&timebuff);
      //calcul timp de start
      startTime=((double)timebuff.time+ ((double)timebuff.millitm)/ (double)1000);
      for (int i=0; i<1000; i++) { Counter->increment(); }          //increment
      //calcul timp stop
      stopTime==((Double)timebuff.time+((double)timebuff.millitm)/((double)1000);
      cout <<(stopTime – startTime) <<”nsecs”<endl; cout <<”Sum =”<<Counter->sum();
      }
catch(CORBA::SystemException &excep)
      {cout<<”System Exception”<<endl;
      cout <<excep;
      return(1);
      }
  return (0);
}
```

Actiunile clientului (count)

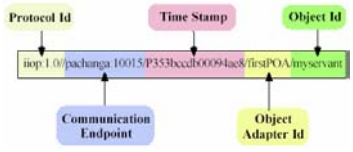


CORBA dinamic



Basic Object Adapter - Metode

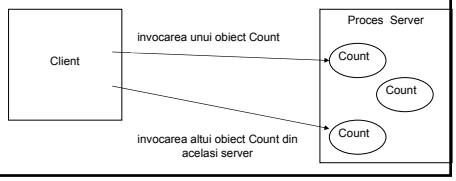
• **create** instanta de obiect si produce o **referintă de obiect** pentru ea.
  • Tipic IOR – Interoperable Object Reference (suporta IIOP)
  • asigura toate info pentru a localiza obiectul



- **change_implementation** actualizarea implementării unui obiect existent.
- **get_id** obtine identificatorul asociat cu obiectul.
- **dispose** distruge referinta obiectului.

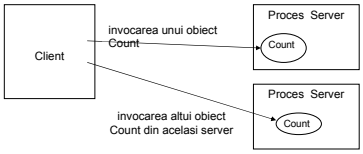
Politici de activare – server partajat

- Un server (proces) contine mai multe obiecte, eventual de clase diferite
- Serverul creează **instante** de obiecte (fabrica de obiecte, constructori)
- Obiectele se înregistrează în BOA.
  - obiect **nou** - invocă **BOA::create** apoi **BOA::Object_is_ready**.
  - obiect **existent** - regăseste si încarcă starea din memoria permanenta. Apoi invocă **obj_is_ready**.
- După pornire obiecte, serverul invocă **impl_is_ready**.
- Un obiect poate fi dezactivat prin **deactivate_obj**.
- Un proces server care se termină anunță BOA prin **deactivate_impl**.



Server nepartajat

- Procesul porneste la invocare obiect
- Dupa initializare, obiectul anunță BOA prin **obj_is_ready**.
- Obiectul rămâne activ până la un apel **dezactivate_obj**.
- La apel obiect inactiv se porneste un nou server (chiar daca e activ alt obiect cu aceeasi implementare).
- Acest mod se foloseste:
  - atunci când obiectele necesită cantități mari de resurse
  - când se doreste mărirea gradului de paralelism (ca alternativă la thread-uri).



Alte moduri

- **Server-per-metodă**
  - Serverul contine codul pentru o singura metoda
  - Serverul este activat odată cu fiecare cerere si dezactivat odată cu satisfacerea cererii.
- **Server persistent**
  - Serverul este activat prin mijloace din afara adaptorului BOA.
  - Odată activat, serverul anunță BOA, printr-un apel **impl_is_ready**, că poate primi cereri de la clienti fiind tratat în continuare ca un server partajat.

POA – Portable Object Adapter

- Propus pentru ca
  - BOA incomplet (ex. interfata de inregistrare a serviciilor cu BOA)
  - a condus la implementari ne-portabile
- Portabilitate realizata prin
  - standardizare clase skeleton produse de compilator IDL
  - standardizare interactiuni intre servicii si adaptorul de obiecte
- Problema
  - Serverele (obiecte) sunt inregistrate cu POA.
  - Clientii detin referinte la obiectele pe care le invoca
  - ORB, POA, serverul si skeleton-ul colaboreaza pentru
    - determinarea operatiei serverului care trebuie invocata
    - realizarea invocarii (dispatch)

"An Overview of the CORBA Portable Object Adapter",
Irfan Pyarali and Douglas C. Schmidt, Department of Computer Science, Washington University

Arhitectura POA

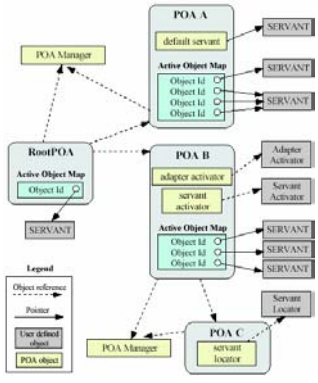
Arhitectura **ierarhica**
- RootPOA
- nested POAs
  - Create de o fabrica din Root
  - Suporta tipuri diferite de obiecte server (ex. tranzitorii, permanente)

Fiecare POA are **Active Object Map**
- Mapeaza **Object Id -> servant** (obiect server)

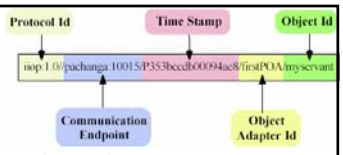
POA manager
- creaza / distruge POA
- activeaza / dezactiveaza POA
- tin / elimina cereri la POA

Adapter Activator
- decide si creaza POA la cerere

Servant Manager (activator, locator)
- activeaza / dezactiveaza servanti
- determina servant exista
- localizeaza servanti

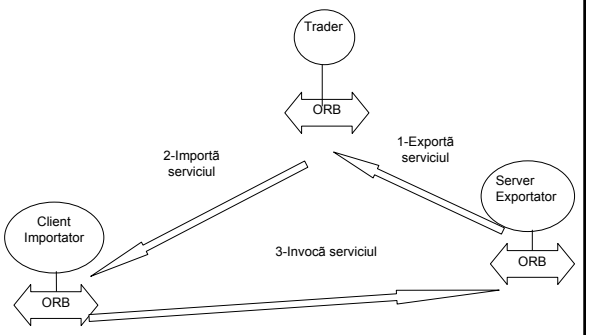


Prelucrarea invocarii

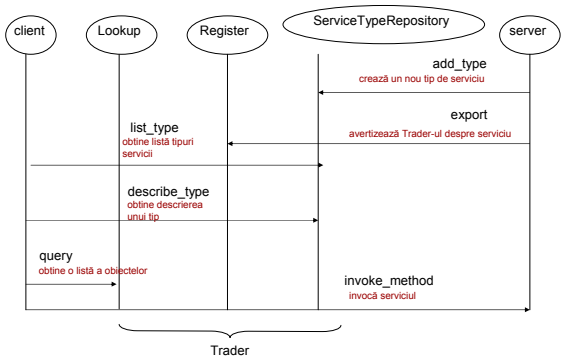


1. ORB localizeaza procesul server folosind Communication Endpoint
   - Foloseste Implementation Repository pentru a crea un nou proces daca este cazul
2. ORB localizeaza POA in procesul server folosind Object Adapter Id
   - Daca e cazul, re-creaza POA (folosind Adapter Activator)
3. ORB livreaza cererea la POA, care localizeaza servant (pe baza Object Id)
4. POA localizeaza skeleton, care "despacheteaza" parametrii (unmarshall) si apeleaza servant
5. Skeleton impacheteaza (marshals) exceptiile, rezultatul, valorile parametrilor inout si out returnate de servant ptr transmitere la client.
   - Alternativ, la exceptie ForwardRequest, determina ORB sa transmita invocare la alt servant.

Serviciul de Trading



Scenariu



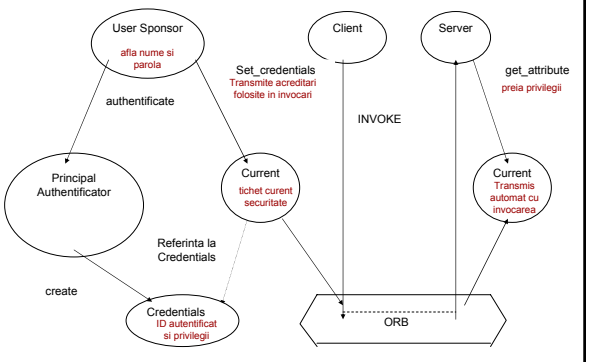
Serviciul de securitate CORBA

- Facilități
  - Autentificarea utilizatorilor
  - Autorizarea accesului la servicii
  - Delegarea de acreditări (sau previlegii)
  - Audit-ul
  - Non-repudierea
  - Criptarea
  - Domenii de încredere
- Interfețe
  - de acces la serviciile de securitate
  - pentru creare şi administrare politici de securitate
  - pentru implementarea serviciilor de securitate

Noțiuni de bază

- **Principal** (protagonist)
  - entitate înregistrată în sistem şi care poate fi autentificată de sistem.
  - Un principal are asociat un set de **acreditări** care determină drepturile sale.
- Acreditările conțin **atributele de securitate,** în două categorii:
  - Atribute de identificare respectiv un **ID autentificat**
  - **Privilegii**
    - ce operații poate face
    - din ce grup face parte
    - ce roluri joacă
    - ce capabilități are.

Autentificarea



Delegarea de acreditări

- **Politici**:
  - fără delegare
  - delegare simplă
  - delegare compusă
- Un client poate **controla** delegarea acreditărilor prin
  - specificarea unui interval de timp sau
  - un număr maxim de invocări pentru care delegarea este validă.
- Un **obiect "conştient"** de serviciul de securitate poate manevra delegarea prin invocarea unor metode ale obiectului current:
  - *get_credentials* întoarce privilegiile proprii
  - *received _credentials* întoarce privilegiile clientului
  - *set_credentials* adaugă privilegiile primite la propriile privilegii

Autorizarea accesului

- Decizia se bazează pe:
  - **privilegiile** curente ale apelantului (proprii sau delegate):
    - identitatea principalului
    - rolurile sale
    - grupurile la care este afiliat
    - verificarea de securitate (security clearance) etc.
  - **Verificări** interval de timp sau numărul de invocări pentru care sunt valabile
  - **Operația** invocată
  - **Politica** de acces a obiectului țintă, ACLs (Access Control Lists).
- **Unitatea de control**
  - Uzual, obiectul.
  - metodele unui obiect
  - colecții de obiecte.
- **Controlul**
  - Explicit: de către server
  - Standard: de către ORB.

Audit-ul

Realizează înregistrarea şi monitorizarea evenimentelor ORB pentru detectare intruşi neautorizaţi.

- **Categorii** de evenimente:
  - **de sistem**: autentificare principal, actualizare privilegii, invocări nereuşite de obiecte;
  - **de aplicaţii**: ex. efectuare plata cu carte de credit.
- Alegerea **politicii de audit**:
  - după obiect sau tip de obiect
  - după timp
  - după atributele principal-ului (ID, rol, etc.)
  - succesul sau nereuşita unei operaţii
- Facilităţi de audit CORBA.
  - La producerea unui eveniment, aplicaţia:
    - obţine referinţa unui obiect **Audit Decision** (invocând **get_policy** pe obiectul **current**)
    - verifică dacă evenimetul trebuie înregistrat (invocând **audit_needed** pe obiectul **Audit Decision**); dacă da, atunci
    - înregistrează evenimetul (invocând **audit_write** pe un obiect **Audit Channel**)
- Evenimentele de sistem sunt înregistrate automat de ORB, în conformitate cu politica de audit stabilită.

Non-repudierea

- Se bazează pe
  - utilizarea unor **certificate de securitate**
  - existența unei autorități de încredere
  - existența unui serviciu de înregistrare a dovezilor.
- Include:
  - evidența creării unui mesaj
    - Transmițătorul creaza un **certificat de origine**, pe care îl transmite odată cu mesajul folosind o **autoritate de livrare**.
    - Receptorul memorează certificatul de origine, folosind un serviciu de **înregistrare şi regăsire a dovezilor**. În cazul unei dispute, receptorul poate regăsi dovada.
  - evidența recepției mesajelor
    - Receptorul creaza si transmite un **certificat de recepție**.
    - Transmițătorul il memorează folosind **serviciul de inregistrare** şi regăsire a dovezilor. In cazul unei dispute, transmițătorul poate regăsi dovada.
  - generarea unor “amprente de timp” care sunt incluse în certificate
  - arbitrarea disputelor de către un judecător care foloseşte dovezile memorate

Domenii de securitate

- Un **domeniu de securitate** este o colecție de obiecte care
  - se supun aceleiaşi politici de securitate,
  - administrată de o aceeaşi autoritate de securitate.
- Domeniul poate avea **subdomenii**.
- Mai multe domenii pot forma o **federație**.
- Un **domeniu de încredre** este alcătuit din obiecte care-şi acordă reciproc încrederea.
- În interiorul domeniului, cerințe de securitate relaxate => eficienta.
- **Inter-operarea** obiectelor din domenii diferite se bazează pe:
  - includerea cerințelor şi mecanismelor de securitate ale unui obiect în **referința de obiect interoperabil** – IOR
  - transmiterea contextului de securitate în cadrul **mesajelor IIOP**

Protocoale Inter-ORB

- CORBA 2.0 defineşte două moduri de inter-operabilitate
  - directă
    - posibilă între două ORB-uri din acelaşi domeniu
      - înțeleg aceleaşi referințe de obiecte
      - acelaşi sistem de tipuri IDL
      - partajează aceeşi informație de securitate.
  - bazată pe o punte (bridge)
    - folosita pentru ORB-uri din domenii diferite.
- Inter-operabilitatea se bazează pe GIOP – General Inter-ORB Protocol
  - specifică sintaxa de transfer şi un set standard de formate de mesaje pentru inter-operarea ORB peste o legătură de transport orientată pe conexiuni.
- IIOP – Internet Inter-ORB Protocol descrie construcția GIOP pe legături de transport TCP/IP.
- IOR – Interoperable Object Reference descrie un format standard de referinte la obiecte

CORBA pentru aplicatii slab-conectate

- CORBA folosita cu succes in multe aplicatii distribuite critice si cu performante ridicate
- WSDL are avantaje certe
  - Clientii si serverele pot
    - "vorbi" mai multe protocoale,
    - folosi diferite formate de mesaje,
    - peste diferite conexiuni de transport, fara porti, punti si intermediari
  - O descriere WSDL expune un contract logic, calitatea serviciului, politicile si legarile fizice pe care le poate oferi
  - Clientul alege legarea convenabila pentru a invoca serviciul
- CORBA binding pentru WSDL
  - Servicii existente (ex. implementate in CORBA) pot deveni parti ale unor aplicatii mai mari, fara modificari, inlocuiri, opriri si reporniri
  - Clientii folosesc specificatii WSDL

"CORBA in a loosely coupled world" by Steve Vinosk, September 8th, 2004

IDL vs. WSDL

Descriere IDL serviciu hello

```
// hello Interface with the greet method
interface hello {
string greet(in string message);
};
```

Descriere WSDL (generata de un translator IDL->WSDL)

Header

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
File generated using SANKHYA Varadhi IDL Compiler
idlc - 1.2 Beta
Language Option: IDL to WSDL Mapping
Please DO NOT edit
-->
```


Definitii – specifica **numele** serviciului si declara **namespace**-uri folosite (referintele la CORBA namespace se datoreaza translatorului)

```
<definitions name="hello"
xmlns:tns="http://www.omg.org/IDL-MAPPED/"
targetNamespace="http://www.omg.org/IDL-MAPPED/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:CORBA="http://www.omg.org/IDL-WSDL/1.0/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<wsdl:documentation>
<CORBA:SourceIDL>
<CORBA:source>hello.idl</CORBA:source>
<CORBA:version>1.0</CORBA:version>
</CORBA:SourceIDL>
</wsdl:documentation>
```

Definitii de tip – independente de masina si limbaj

```
<types>
<xsd:schema targetNamespace="http://www.omg.org/IDL-WSDL/1.0"
xmlns="http://www.w3.org/2001/XMLSchema">
<xsd:simpleType name="CORBA.completion_status">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="COMPLETED_YES"/>
<xsd:enumeration value="COMPLETED_NO"/>
<xsd:enumeration value="COMPLETED_MAYBE"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="CORBA.SystemException">
<xsd:sequence>
<xsd:element
name="minor" type="xsd:unsignedInt"
maxOccurs="1" minOccurs="1"/>
<xsd:element
name="completion_status" type="CORBA.completion_status"
maxOccurs="1" minOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</types>
```

Mesaje – folosite in implementarea **operatiilor**

```
<!-- Message related to CORBA System Exception-->
<message name="CORBA.SystemExceptionMessage">
<part name="_return" type="CORBA.SystemException"/>
</message>
<!-- Messages related to portType : hello -->
<message name="hello.greet">
<part name="message" type="xsd:string"/>
</message>
<message name="hello.greetResponse">
<part name="_return" type="xsd:string"/>
</message>
```

Port type – defineste o **interfata** cu zero sau mai multe **operatii**

```
<!-- portType for interface "hello" -->
<portType name="hello">
<operation name="greet">
<input message ="tns:hello.greet"/>
<output message ="tns:hello.greetResponse"/>
<fault message ="tns:CORBA.SystemException"/>
</operation>
</portType>
```

Binding information (nu apar in CORBA) – specifica legarile pentru fiecare operatie: protocolul de mesagerie, modelul de codificare date, transportul folosit

```
<!-- binding for interface "hello" -->
<binding name ="helloBinding" type="tns:hello">
<soap:binding
style="rpc"
transport = "http://schemas.xmlsoap.org/soap/http"/>
<operation name="greet">
<soap:operation soapAction="hello#greet"/>
<input>
<soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.omg.org/IDL-MAPPED/hello"
use="encoded"/>
</input>
<fault message ="CORBA.SystemException"/>
</operation>
</binding>
```

Service – specifica adresa pentru invocarea serviciului

```
<service name ="helloService">
<port name ="helloPort" binding="tns:helloBinding">
<soap:address location="http://www.dummyurl.com/hello"/>
</port>
</service>
</definitions>
```

Comparatie

IDL	WSDL
type	type
method	operation
interface	portType
object IOR	service