Sisteme distribuite – Tehnologii

Sisteme distribuite bazate pe obiecte

Modelul arhitectural cu doua nivele

- A dominat perioada timpurie a arhitecturilor sistemelor distribuite
- Arhitectura client-server
  - Primul nivel (superior) trateaza prezentarea si logica afacerii aplicatiei utilizator (client),
  - Al doilea nivel (inferior) trateaza organizarea aplicatiei si stocarea datelor sale (server).
- Intensiv utilizata in
  - Planificarea resurselor intreprinderilor
  - Plati
  - Sisteme applicative de inventariere (aplicatii de afaceri tip client care localizate in sisteme desktop multiple interactioneaza cu un server central).

Limitarile de baza ale modelului aplicatiilor client-server

- Procesarea complexa la partea client necesita ***sisteme client robuste.***
- ***Securitatea*** este dificil a fi implementata deoarece algoritmii si logica rezidenta la partea client o face vulnerabila hackerilor
- O latime de banda considerabila este necesara pentru a raspunde la apeluri numeroase este necesara la partea serverului, ceea ce impune restrictii de ***scalabilitate***.
- ***Mentinerea si actualizarile*** aplicatiilor client sunt extrem de dificile deoarece fiecare client trebuie mentinut separat.
- Arhitectura client-server este adecvata mai ales aplicatiilor de sien-statatoare orientate spre baze de date si ***nu tintesc*** aplicatii robuste bazate de componente reutilizabile.

CORBA

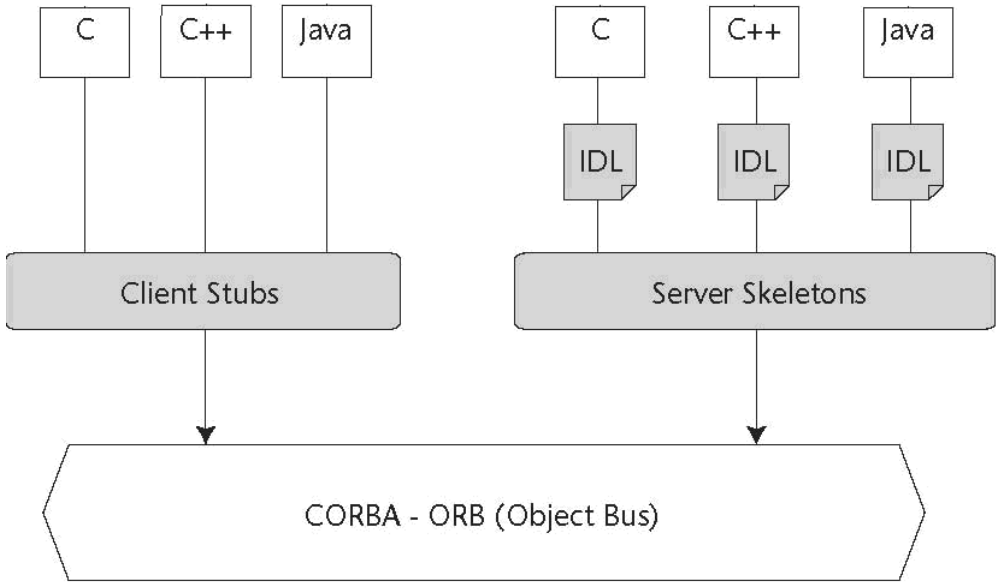
- Common Object Request Broker Architecture
- utilizat in industrie pe scara larga, initiativa de standardizare deschisa, dezvoltata de Object Management Group (OMG)
- difera de modelul traditional client-server:
  - ofera o solutie orientata spre obiecte care nu forteaza nici un protocol particular si nici un limbaj de programare particular, sistem de operare sau platforma hardware
- Interface Definition Language (IDL) este un limbaj specific pentru interfete construit pentru a servicii (metode/functii) ale obiectelor CORBA la distanta.
- defineste o colectie de servicii la nivel de sistem pentru tratarea serviciilor aplicatiilor de nivel de baza precum ciclul de viata, persistenta, trazactie, numire, securitate, etc

ORB

- Object Request Broker
- Este o magistrala pentru obiecte care ofera un mecanism transparent pentru expedierea cererilor si receptionarea raspunsurilor catre si de la obiecte, indiferent de mediu si locatia lor.
- Intercepteaza apelurile clientului si este responsabil pentru gasirea obiectului server care implementeaza cererea, paseaza paraetrii, invoca metoda si returneaza rezultatele la client.
- Ofera interfete catre serviciile CORBA care-l permit construirea de medii de aplicatii distribuite customizate.
- CORBA 2.0 adauga interoperabilitate intre diferiti producatori ORB prin implementarea unui protocol Internet Inter-ORB Protocol (IIOP):
  - IIOP defineste coloana de sustinere ORB, prin care fiecare ORB poate lega si oferi inter-operatii ale serviciilor sale asociate

Modelul arhitectural CORBA

- Contractele IDL specifica domeniul aplicatiei si stabiliste interfetele cu clientii sai
- ORB se comporta ca o magistrala pentru obiecte sau punte de legatur ce ofera infrastructura de comunicare pentru a trimite si primi cereri/raspunsuri de la client si server



Avantajele CORBA

- Independenta de sistemul de operare si de limbajul de programare
- Integrarea apliactiilor existente
- Infrastructura bogata in obiecte distribuite
- Transparenta locatiei.
- Transparenta retelei.
- Interfata de invocare dinamica

Dezavantajele CORBA

- ***Investitie initiala ridicata.*** Aplicatiile bazate pe CORBA necesita investitii enorme in ceea ce priveste trainingul si lansarea arhitecturii, chiar si pentru aplicatii de dimensiuni mici.
- ***Disponibilitatea serviciilor CORBA.*** Serviciile obiect specificate de OMG lipsesc inca in produse care implementeaza conceptele.
- ***Scalabilitatea.*** Datorita naturii strans cuplate a arhitecturii CORBA orientate pe legaturi, scalabilitatea inalta asteptata in aplicatiile companiilor poate sa nu fie atinsa.

Java RMI

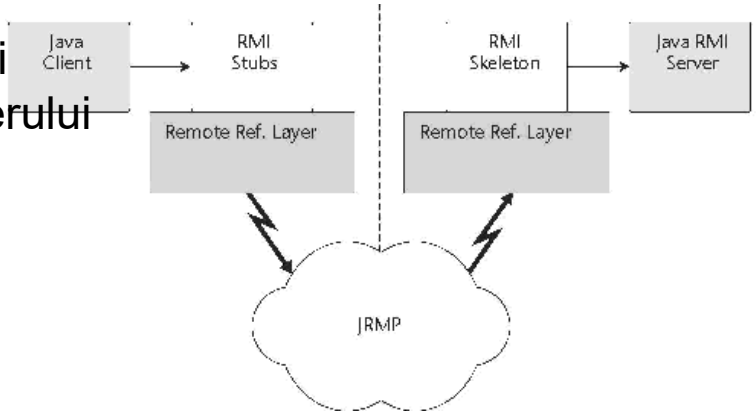
- Permite calcul distribuit orinetat spre obiecte
- Dezvoltat de Sun Microsystems ca si mecanism standard pentru a permite dezvoltarea de aplicatii disribuite Java bazate pe obiecte
- Permite apelul obiectelor Java aflate la distanta si pasarea lor ca argumente sau valori la returnare.
- Utilizarea serializarea obiectelor Java object—technica de persistenta a obiectelor care permite conversia obiectelor in streamuri.

Modelul arhitectural RMI (1)

- Un mecanism orientat spre registru (rmiregistry) ofera un servicii simplu ne-persintent de cautare de nume care este utilizat pentru a stoca referinte la obiecte la distanta si pentru a permite cautarea de catre aplicatii client
- Java Remote Method Protocol (JRMP) este un protocol de comunicare intre procese ce permite obiectelor Java care exista in diferite masini virtuale Java sa-si invoce transparent metodele unele alteia
- Un mecanism de colectoare a gunoaielor prin numerotarea referintelor tine evidenta referintelor obiectelor existente la obecte la distanta (conexiuni vii) utilizand masina virtuala

Modelul arhitectural RMI (2)

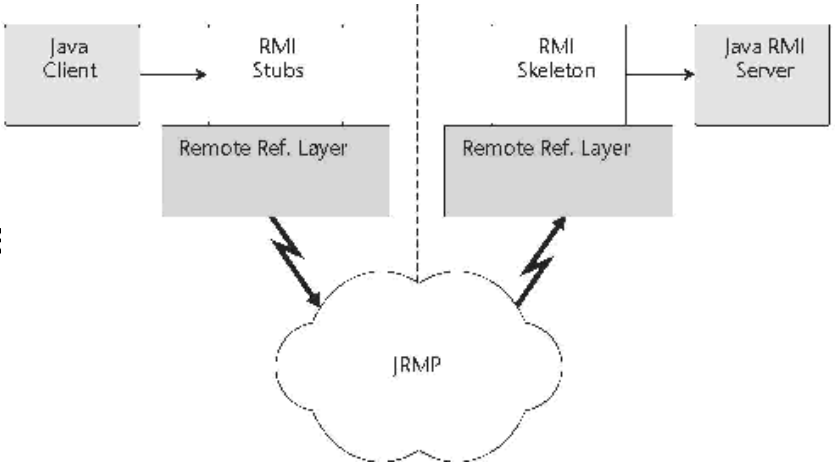
- Client RMI: applet Java / aplicatie de sine-statatoare,
  - Efectuaza invocarea metodelor la distanta a obiectului server;
  - Paseara argumentele care au tipuri de date primitive sau sunt obiecte serializabile.
- Stub RMI: proxy client
  - Generat de compilatorul RMI (*rmic* in JDK)
  - Incapsuleaza informatiile de retea ale serverului
  - Efectueaza delegarea metodei invocate a serverului
  - Pune in sir (marshal) argumentele metodei
  - Scoate din sir (unmarshal) valorile de returnare de la executia metodei.



- Infrastructura RMI are doua straturi:
  - Stratul referintei la distanta
    - Separa comportarea referintei la distanta sepcifice de stub-ul client
    - Trateaza anumita semantica a referintei precum reluarea conexiunii, care sunt cereri de invocare unicast care multicast;
  - Startul de transport facilitieaza
    - Transferul efectiv a datelor la invocarea metodelor,
    - Pasarea argumentelor formale si
    - Returnarea rezultatelor executiei.

Modelul arhitectural RMI (3)

- Scheletul RMI (skeleton):
  - generat utilizand compilatorul RMI (rmic)
  - receptioneaza cererile de invocare de la stub
  - proceseaza argumentele (unmarshal)
  - le deleaga la serverul RMI
  - pune in sir (marshal) valorile de returnare si le paseaza la stubul RMI stub prin infrastructura RMI
- Serverul RMI: obiectul Java la distanta care
  - Implementeaza interfetele expuse si executa cererile clientilor
  - Receptioneaza invocarile metodelor la distanta de la scheletul corespunzator care-l paseaza parametrii dupa scoaterea din sir
  - Returneaza valori care sunt transmise la schelet care le paseaza la client prin infrastructura RMI.



Avantajele Java RMI

- Dezvoltarea aplicatiilor distribuite in RMI este mai simpla decat dezvoltarea cu socluri Java: nu este necesara construirea unui protocol
- RMI este construit peste socluri TCP/IP, dar avantajul in plus este acela de a oferi orientare obiect pentru comunicare inter-procese
- Ofera un mecanism eficient si tranparent de comunicare care elibereaza programatorii de toate protocoalele necesare la nivelul aplicatiei pentru codarea si decodarea mesajelor pentru schimbul de mesaje
- RMI permite administrarea distribuita a resurselor, utilizarea eficieta a puterii de calcil si balansarea incarcarii intr-un model al aplicatiilor Java
- RMI-IIOP (RMI over IIOP) este un protocol dezvoltat pentru a pernite applicatiilor RMI sa inter-opereze cu componente CORBA.

Dezavantaje ale Java RMI

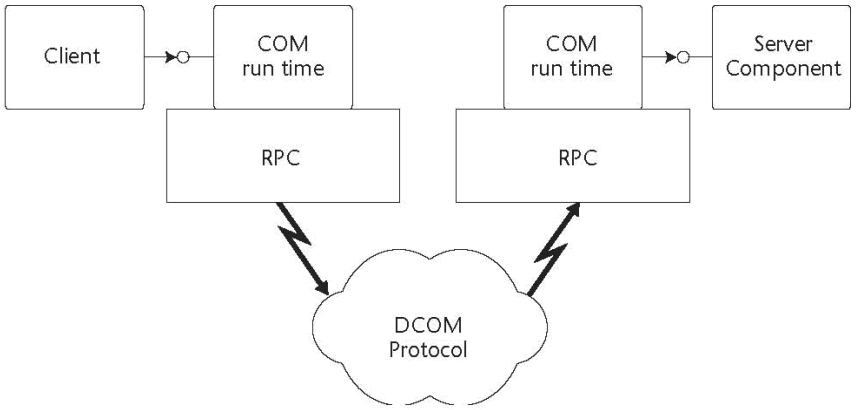
- RMI este limitat numai la platforma Java.
  - Nu ofera independenta de limbaj in modelul sau distribuit precum este scopul in cazul CORBA.
- Arhitectura aplicatiilor bazate pe RMI este strans cuplata datorita naturii orientate pe legatura
  - Atningerea unei scalaibilitati ridicate intr-un asemenea model de aplicatii este o provocare.
- RMI nu ofera nici un suport pentru administrarea unei sesiuni specifice
  - Intr-o implemetare tipica client/server, serverul mentine sesiunea si informatia de stare asupra clientilor multiplii care il acceseaza
  - Mentinerea unei asemenea informatii intr-o aplicatie service RMI fara un suport standard este o sarcina complexa

Microsoft DCOM

- Microsoft Component Object Model (COM)
- O modalitate pentru componentele bazate pe Windows sa comunice intre ele prin definirea unui standard binar si de retea in mediul de operare Windows
- Permite aplicatiilor COM sa comunice intre ele utilizand un mecanism RPC, ca utilizeaza un protocol DCOM

Modelul arhitectural al Microsoft DCOM

- Abordare skeleton si stub
- Stub
  - Incapsuleaza informatia de localizare in retea a obiectului server COM
  - Se comporta ca un proxy la partea client
- Serverele
  - Pot gazdui potential obiecte multiple COM
  - Se inregistreaza singure la un registru
- Clientii descopera serverele printr-un mecanism de cautare (lookup)



Avantajele si dezavantajele DCOM

- Are succes in oferirea de suport pentru calcul distribuit pe platforme Windows
- Limitarile comune pentru DCOM:
  - Dependenta de platforma: limitat la mediul aplicatiilor Microsoft
  - Probleme la managementul starii
  - Probleme cu scalabilitatea
  - Probleme legate de administrarea sesiunilor complexe

Middleware orientat spre mesaje (MOM)

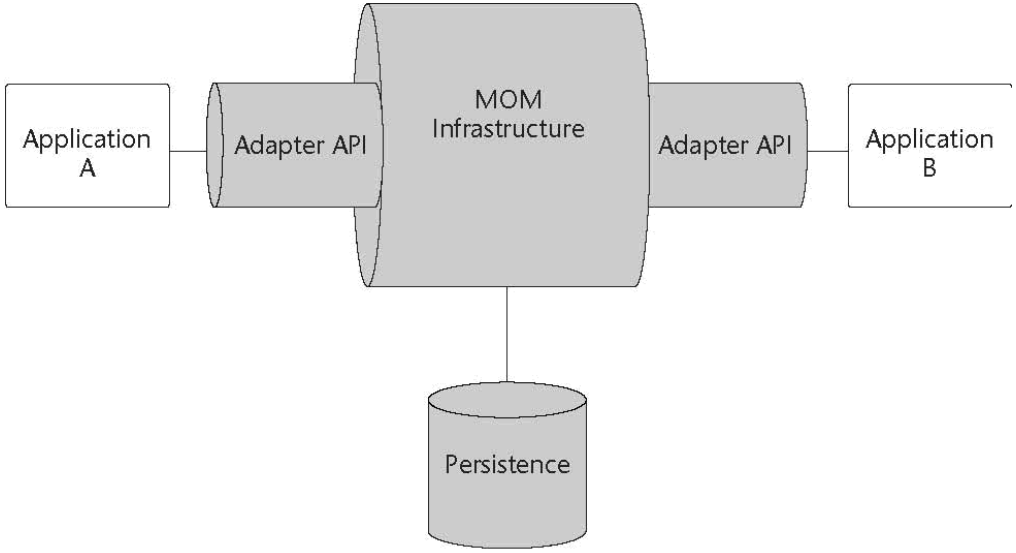
- CORBA, RMI si DCOM au adoptat un mecanism de cuplare stransa legat printr-un model de comunicare sincron (cerere/raspuns).
- MOM este bazat pe un model de comunicare asincrona cu cuplare slaba
  - Clientul aplicatiei nu trebuie sa stie recipientii aplicatiei si argumentele metodelor sale
- Permite aplicatiilor sa comunice indirect utilizand o coada de mesaje la furnizor
  - Clientul aplicatiei expediaza mesaje la coada de mesaje (o arie de mentinere a mesajelor) si
  - Aplicatia receptor preia mesajul din coada
  - Aplicatia ce expediaza mesajele la o alta aplicatie continua sa opereze fara a astepta raspunsul de la acea aplicatie

Provocari comune in calculul distribuit

- Context:
  - CORBA, RMI si DCOM au avut succes in integrarea aplicatiilor intr-un mediu omogen intr-o LAN
  - O scara la nivel Internet necesita interoperabilitatea aplicatiilor peste retele
- Mentinerea unor versiuni numeroase a stub/skeleton in mediile clientului si serverului este extrem de complexa in medii eterogene bazate pe retea
- Tinte ale calitatii serviciilor (QoS) precum Scalabilitatea, Performanta, si Disponibilitatea intr-un mediu distribuit consuma o buna bucata din timpul de dezvoltare a aplicatiei
- Interoperabilitatea aplicatiilor ce implementeaza diferite protocoale peste platforme eterogene este aproape imposibila
  - Ex. Unui client DCOM comunicand cu un server RMI sau un client RMI comunicand cu un server DCOM.
- Majoritatea acestor protocoale sunt construite pentru a opera cu succes in retele locale.
  - Nu sunt prietenoase cu firewall friendly sau capabile sa fie accesate din Internet.

Modelul arhitectural MOM

- Aplicatiile interactioneaza cu infrastructura de transmitere de mesaje utilizand adaptori customizati
- Pentru o livrare de incredere a mesajelor, mesajele pot fi retinute si intr-o baza de date sau sistem de fisiere



Implementari si limitari ale MOM

- SunONE Message Queue, IBM MQSeries, TIBCO, SonicMQ, Microsoft Messaging Queue (MSMQ).
- JMS: Java Message Service, este dezvoltat ca parte a Sun Java Community Process (JCP) si este parte din J2EE
  - JMS ofera modele de transmitere de mesaje punct-la-punct si publicare/subscriere cu urmatoarele facilitati: securitate, livrarea de incredere a mesajelor si facilitati complete pentru tranzactii
- Provocarile comune in implementarea unui mediu pentru aplicatii bazate pe MOM:
  - Majoritatea implementarilor standard MOM au oferit APIuri native pentru comunicarea infrastructurii lor de baza
    - aceasta afecteaza portabilitatea aplicatiilor intre aceste implementari si conduce la blocare la un producator specific
  - Mesajele MOM utilizate pentru integrarea aplicatiilor sunt uzual bazate pe formate de mesaje proprietar fara a se supune nici unui standard