Sisteme distribuite – Teorie
2. Design si middleware

Software cuplat slab vs. cuplat strans

Cuplat slab:

- Permite masinilor & utilizatorilor unui DS sa fie independent unul de altul
- Exemple:
  1. PCuri care partajeaza anumite resurse, precum imprimante sau baze de date, peste o LAN
  2. Sisteme de operare de retea: sistem de fisiere partajat, fiecare calculator are sistemul sau de operare si se supune cererilor utilizatorului

Strans cuplat:

- Sistem de partajare a timpului care este unic
- Utilizatorii nu sunt consienti de existenta mai multor procesoare in sistem
- Exemple:
  - Sistemele cluster

Administrarea proceselor si sistemelor de fisiere intr-un SD

- Mecanism global unic pentru comunicare intre procese a.i. orice proces poate discuta cu oricare alt proces
- Modalitatea in care procesele sunt create, distruse, pornite si oprite nu trebuie sa varieze de la masina la masina
- Sistemul de fisiere trebuie sa arate la fel de oriunde (sistem global de fisiere)
- Oriunde aceeasi interfata de apel de sistem => (?) nuclee identice care sa ruleze pe toate masinile sist.
- Cand un proces trebuie pornit, toate nucleele trebuie sa coopereze pt. a gasi cel mia bun loc pentru executia sa

Problema de design: 1. Transparenta

- SD: o multime de procese cooperante
- Complexitatea rezulta din faptul ca distributia trebuie sa fie realizata transparent (mai exact invizibila) pentru programatorul aplicatiilor
- Un sistem cu partajarea timpului care ofera o imagine a unui sistem singular se considera a fi transparent
- Exemple:

1. Utilizatorul unei comenzi make din Unix pentru a compila un numar mare de fisiere nu trebuie sa stie ca toate compilatoarele actioneaza in paralel pe masini diferite
2. Citirea unor fisiere la distanta in aceeasi maniera ca si in cazul fisierelor locale

Tipuri de transparenta

Tip	*Semnificatie*
Transparenta locatiei	Utilizatorul nu poate spune unde sunt localizate resursele
Transparenta migrarii	Resursele pot fi mutate oricand fara a le schimba numele
Transparenta replicarii	Utilizatorul nu poate spune cate copii exista
Transparenta concurentei	Utilizatorii multiplii pot partaja automat resurse
Transparenta paralelismului	Se pot executa activitati in paralel fara ca utilizatorii sa stie
Transparenta accesului	Modalitati identice in care accesul are loc la componente locale sau la distanta
Transparenta esecurilor	Utilizatorii nu sunt constient de esuarea unui componente
Transparenta tehnologiei	Tehnologii diferite, precum limbajele de programare sau sistemele de operare sunt ascunse de utilizator

Problema de design: 2. Flexibilitatea

- Utilizarea de sisteme monolotice sau micro-nuclee?
- Utilizarea sistemelului monolotic: un sistem centralizat + facilitati de retea
- Micro-nucleele sunt mai flexibile deoarece nu fac mare lucru

Problema de design: 3. Incredere

- Unul dintre scopurile SD: sa se realizeze un sistem mai de incredere decat un sisteme bazat pe un singur procesor
- Exemplu : daca o masina nu mai este disponibila, altele ii preiau sarcinile
- OR boolean a increderii in componente: daca o masina are probabilitatea sa fie disponibila de 95%, posibilitatea ca patru sisteme dintr-un SD sunt indisponibile este de (5%)^4=0.0006 % <<5%
- Zicala: “definitia” Lamport a unui SD: este un sistem “in care nu putem obtine rezultatele dorite pentru ca anumita masina despre care n-am auzit niciodata a cazut"

Aspectele increderii

- Disponibilitatea: fractia de timp in care sistemul este utilizabil – poate fi imbunatatita prin:
  - Un design care nu necesita functionarea simultana a unui numar substantial de componente critice
  - Redundanta: piesele cheie hardware si software trebuie replicate, astfel incat daca una dintre ele cade celelalte preiau sarcinile
    - Mai multe copii -> disponibilitate mai buna -> sansa mai mare pentru inconsistenta copiilor
- Securitate – resursele protejate la utilizare neautorizata (mai sever in SD decat intr-un sistem uni-procesor)
- Toleranta la erori
  - precum efectele unui reboot neplanificat a unui server
  - SD trebuie proiectat pentru a masca erorile: ascunderea acestora fata de utilizatori

Probleme de design: 4. Performanta

- Rularea unei aplicatii intr-un SD n-ar trebui sa se comporte mai prost decat la rularea unei aplicatii pe un singur procesor
- Metrici:
  - Timpul de raspuns
  - Throughput – numarul de joburi per ora
  - Utilizarea sistemului
  - Cantitate de capacitate a retelei care a fost consumata
- Probleme de performanta: coomunicarea este de obicei inceata=> necesara minimizarea numarului de mesaje
- ? Calea cea mai buna: performanta se castiga prin faptul ca mai multe procese ruleaza in paralel pe masini diferite, dar aceasta inseamna si faptul ca se trimit mesaje multiple
  - Un sinugur calcul la distanta, precum adunarea a doi intregi, nu este indicata
  - O sarcina de calcul de lunga durata la distanta este mai adecvata

Vezi cursul de calcul paralel din Sem.II!

Probleme de design: 5. Scalabilitatea

- In mod uzual un SD are cateva sute de CPUri.
- Gridurile: cateva mii
- Sistemele de rezervare electronica pot avea zeci de milioane
- Problema: solutia care functioneaza acceptabil pentru 200 de masini poate lucra mizerabil pentru 200,000,000
- Principiu de ghidare:
  - Evitarea componentelor centralizate
    - Contra-exemplu: un singur server de e-mail pentru 50 milioane de utilizatori (ne-tolerant la erori, congestie de retea etc)
  - Evitarea tabelelor centralizate
    - Contra-exemplu: o singura baza de date tine evidenta numerelor de telefon si adresele a 50 milioane de oameni
  - Evitarea algoritmilor centralizati
    - Contra-exemplu: cazul unui SD mare cu numeroase mesaje care trebuie rutate pe diverse linii
      - Imagine gresita: modalitatea ideala este aceea de a colecta informatii complete la un server despre incarcarile tuturor masinilor si liniilor, si rularea unui algoritm bazate pe teoria grafurilor (la server) pentru a calcula toate drumurile optime; trimite apoi informatia in sistem pentru a imbunatati rutarea

Algoritmi descentralizati - caracteristici

1. Nici o masina nu are informatia completa asupra starii sistemului
2. Masinile iau decizii bazate numai pe informatiile locale
3. Esecul unei masini nu ruineaza algoritmul
4. Nu exista o presupunere implicita ca exista un ceas global
   - Contra-exemplu: Orice algoritm care porneste cu “La 12:00:00 fix toate masinile trebuie sa noteze dimensiunea cozii de iesire” va esua deoarece este imposibil sa obtii toate ceasurile sincronizate exact

Middleware

- Ofera servicii generale care permit executia distribuita a aplicatiilor
- Este software-ul pozitionat intre sistemul de operare si aplicatie
- O “fata-de-masa” care se intinde peste o retea eterogena, ascunzand complexitatea tehnologiilor care permit aplicatiilor sa fie rulate in respectivul mediu

Sarcinile middleware-ului in cazul unei aplicatii OO (1)

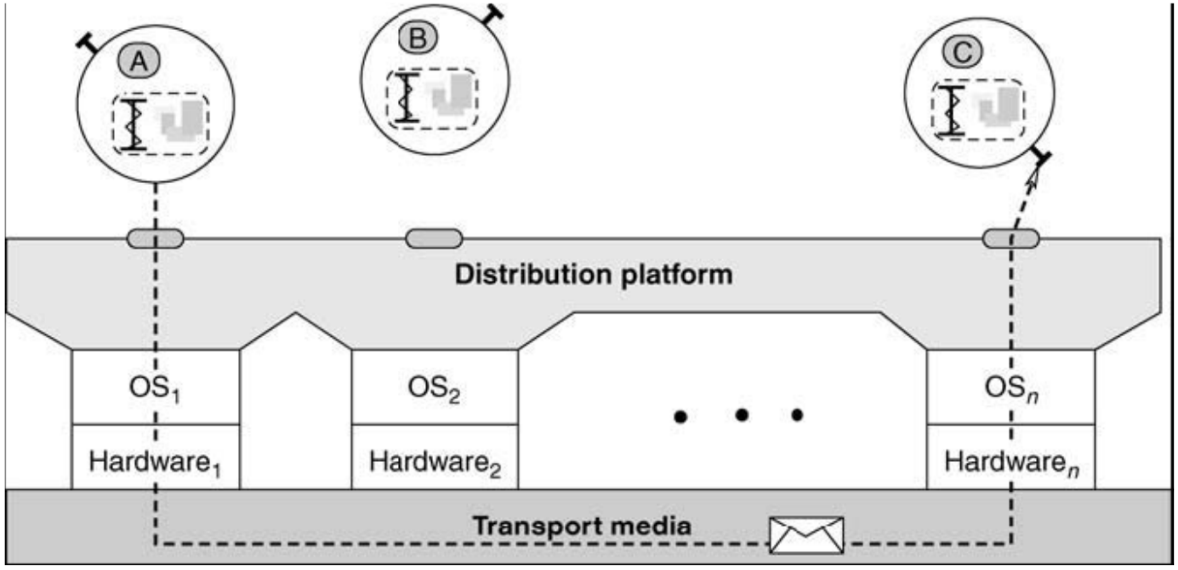
- Obiect:
  - Incapsuleaza starea si comportarea si poate fi accesat numai via o interfata bine definita
  - Interfata ascunde detaliile care sunt specifice implementarii, si astfel ajuta la incapsularea tehnologiilor diferite
  - Este o unitate pentru distribuitie
  - Comunica intre ele prin schimb de mesaje

Sarcinile middleware-ului in cazul unei aplicatii OO (2)

- Suport pentru modelul obiectual:
  - Middleware-ul trebuie sa ofere mecanismele care sa suporte conceptele incorporate in modelul obiect
- Interactiunea operationala:
  - Middleware ar trebui sa permita interactiunea operationala intre doua obiecte
  - Modelul utilizat consta in invocarea metodei intr-un limbaj OOP
- Interactiunea la distanta:
  - Middleware ar trebui sa permita interactiunea intre doua obiecte localizate in spatii de adresare diferite
- Transparenta distributiei:
  - Din punctul de vedere al programului, interactiunea intre obiecte este identica pentru interactiunile locale ca si pentru interactiunile la distanta
- Independenta tehnologica:
  - Middleware-ul permite integrarea unor tehnologii diferite

Structura unei platforme middleware

- Middleware-ul este localizat conceptual intre aplicatie si sistemul de operare
- Aplicatia este reprezentata de o multime de obiecte care interactioneaza
- Fiecare obiect este explicit alocat la o platforma hardware



Middleware-ul ascunde eterogeneitatea

Eterogeneitatea exista in locuri diferite:

- Limbaje de programare:
  - Obiecte diferite pot fi dezvoltate in diferite limbaje de programare
- Sisteme de operare:
  - Sistemele de operare pot avea caracteristici si facilitati diferite
- Arhitectura calculatoarelor:
  - Calculatoarele difera in detaliile lor tehnice (exemplu, reprezentarea datelor).
- Retele:
  - Calculatoare diferite sunt legate intre ele prin diferite tehnologii de retea

Cum trateaza middleware-ul eterogeneitatea

- Ofera acceasi functionalitate la toate punctele de acces:
  - Aplicatiile au access la functionalitatea lor printr-un API; APIurile sunt adaptate la conditiile fiecarui limbaj de programare care este suportat de catre middleware.
- Un programator de aplicatii in mod tipic vede middleware-ul ca o biblioteca de programe sau ca un set de unelte
  - Dependent de mediul de dezvoltare pe care programatorul il utilizeaza
  - Afectat de asemenea de compilatorul/interpretorul utilozat pentru dezvoltarea aplicatiei distribuite

Standardizarea middleware-ului

- Cand se proiectaeaza un middleware pentru o retea globala, la scara multinationala, se remarca caracteristici speciale care difera de cele ale sistemelor distribuite restrictionate geografic
  - Middleware-ul cuprinde mai multe tehnologii si domenii politice (in send de securitate)
  - Nu se poate presupune ca exista o tehnologie omogena in sistemul distribuit
- Nu se poate presupune ca un singur vendor este capabil sa ofere middleware sub forma de produse pentru toate mediile
  - evitarea monopolului!
  - Inovare prin competitie
  - implementarea middleware-ului prin mai multe produse care concureaza pote conduce la solutii partiale care nu sunt compatibile
- Compatibilitatea este posibila numai daca toti vendorii de middleware adera la un standard

Standard

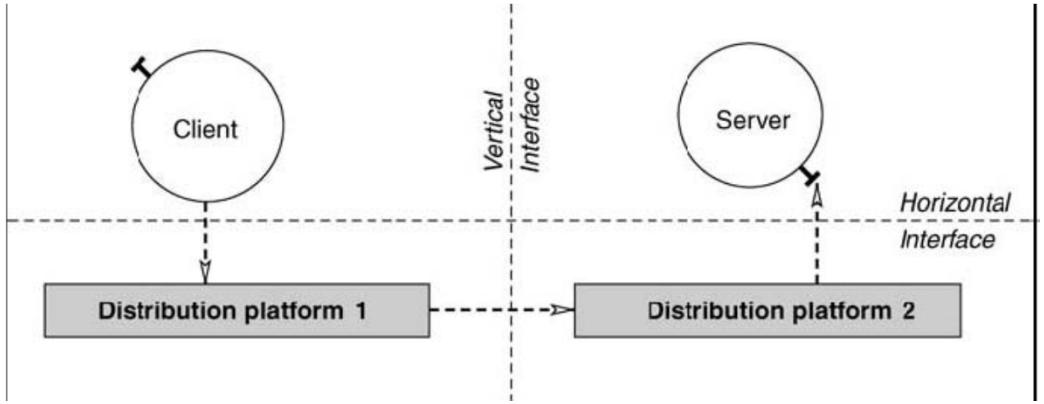
- Stipuleaza *specificatia (descrierea detaliata)* a unui produs —
  - O descriere abstracta a comportarii dorite care permite un grad de libertate in executia unei implementari.
- Servesta ca material de referinta conform caruia pot fi produse diferite implementari
- O specificatie identifica caracteristicile esentiale ale unui sistem
  - Daca un sistem practic se conformeaza standardului, trebuie sa aiba aceste caracteristici
- Avantaje:
  - Garanteaza independenta vendorului, permitand astfel clientului sa selecteze dintr-o varietate de produse fara a adera numai la un vendor particular
  - Protectia investitiei utilizatorului pentru a utiliza un produs

Caracteristicile standardelor deschise

- Non-proprietar:
  - Standard insusi nu este subiect pentru orice interes comercial
- Disponibil gratuit:
  - Acesul la standard este disponibil oricui
- Independent de tehnologie:
  - Standardul reprezinta o abstractizare a mecanismelor tehnice concrete si defineste numai un sistem in masura in care este necesara compatibilitatea intre produse
- Proces de creare democratic
  - Crearea si evolutia ulterioara a standardului nu este dominata de o singura companie, ci are loc printr-un proces democratic
- Disponibilitatea produsului:
  - Un standard este efectic numai daca exista produse pentru el

Interfate intre componentele SD

- In contextul middleware-ului, un standard trebuie sa stabileasca interfetele intre diferite componente pentru a permite interactiunea dintre ele
- Doua tipuri de interfete: orizontale si verticale



Interfata orizontala / API / Portabilitate

- Exista intre o aplicatie si middleware
- Defineste cum o aplicatie poate accesa functionalitatea middleware
- Este de asemenea referit ca un *Application Programming Interface* (API)
- Standardizarea interfetei intre middleware si aplicatiei are ca si consecinta *portabilitatea* aplicatiei in middleware-uri diferite:
  - acelasi API exista la fiecare punct de acces
- Programatorii aplicatiilor sunt interesati de obicei numai in interfata orizontala pentru ca aceasta defineste punctul de contact a aplocatiilor lor

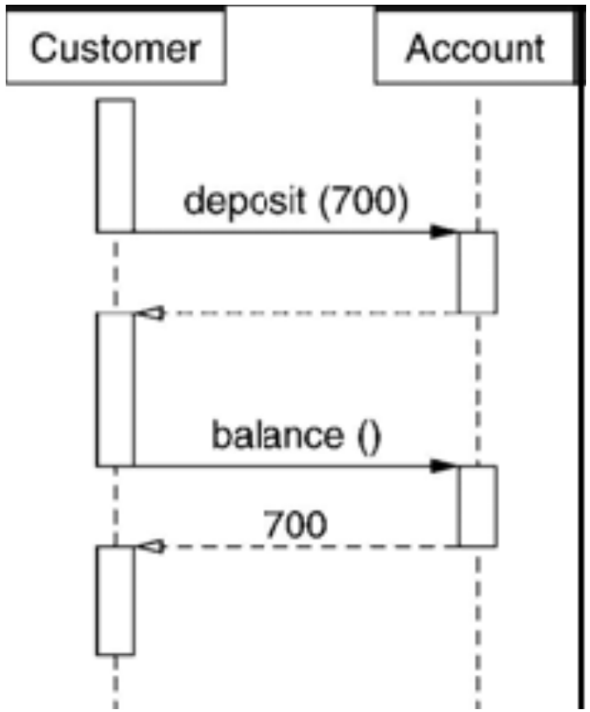
Interfata verticala / Interoperabilitate

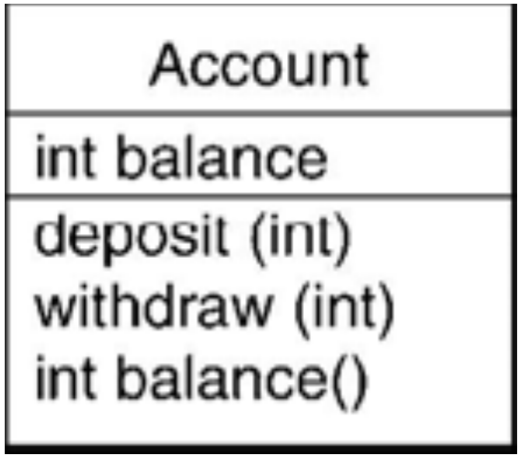
- Defineste interfata intre doua instante ale platformei middleware
- Este in mod tipic definita printr-un proocol bazat pe mesaje, referit ca si *protocol data units* (PDU-uri).
  - Un PDU este un mesaj expediat in retea
  - Atat clientul cat si serverul schimba PDU-uri pentru implementarea protocolului.
- Separa domeniile tehnologice
- Asigura faptul ca aplicatiilor pot fi extinse dincolo de aria de influence a unui produs al middleware-ului
- Standardizarea acestei interfete permite *interoperabilitatea* intre aplicatii
- De importanta minora pentru dezvoltarea unei aplicatii
- Dependenta implicita exista intre interfetele verticale si orizontale
  - De exmplu, regulile de codare pentru PDU-uri trebuie sa existe in interfata verticala pentru toate tipurile de date disponibile in inetrafta orizontala a aplicatiei

Aplicatie pentru exemplificare: cont bancar

- Clientul doreste sa efectueze anumite operatii asupra unui cont bancar
- Nu ne intereseaza diferitele tiprui de conturi, ci cum sunt create conturile de o banca
- Pentru simplicitate, presupunem ca exista numai un singur client si un singur cont, fiecare reprezentat prontr-un obiect
- Contul mentine o balanta
- Clientul poate depozita si retrage bani prin operatiunile corespunzatoare
- Clientul poate cere de asemenea balanta contului

Diangrama secventa pentru cazul utilizarii contului & diagrama de clase in UML





Distribuirea aplicatiei de exemplu

- Nivele:
  - Nivelul server contine obiectu cont.
  - Nivelul client aceseaza acest obiect prin referinte (de exemplu pointeri C++).
- Separarae clientului si serverului in diferite spatii de adresare – se presupune ca:
  - Parametrii curenti sunt trasnmisi intre procese deoarece nu exista un spatiu comu de adrese
  - Toate datele care apartin parametrilor unei interactiuni intre client si server trebuie astfel trasnmise explicit la spatiul de adrese ale serverului
  - Data trebuie sa fie de sine statatoare; adica nu este permisa utilizatea pointerilor care sunt valizi numia in contextul clientului

Proxy

- Un proxy la server exista la partea clientului pentru
  - A oferi acelasi API ca si serverul insusi
  - A transmite toti parametrii printr-un canal de comunicare la spatiul de adrese la distanta
- In spatiu de adresare la distanta, un proxy al clientului
  - Accepta datele si
  - Executa invocarea curenta la server
- Nu este posibila distinctia intre proxy si “originalul” sau, astfel incat distributia clientului si a serverului este *transparenta.*
- Proxy-urile sunt utilizate pentru a umple golul la fiecare parte astfel incat clientul si serverul nu sunt constienti de separare