Minta ZH

Elosztott vs. központosított

Az elosztott rendszerek erőssége a nagy rendelkezésre állás, és az információ valamint az erőforrások nagy földrajzi területen való megosztásának képessége, így nagymértékű szabadságot biztosít a rendszertervezőnek, hogy a rendszer elemek (adatok, feldolgozás) optimális elhelyezésére.

- rugalmasság: A számítógépek és az információs rendszer más elemei (adatok, feldolgozás) a szervezetben oda helyezhetők, ahol a leghatékonyabb a kihasználásuk. A rendszerhez könnyen hozzáadhatók új komponensek, szintén egyszerű a meglévő részek eltávolítása, frissítése, más helyre mozgatása anélkül, hogy ez a rendszer többi részére hatással lenne.
- skálázhatóság: A rendszernek az a tulajdonsága, hogy kapacitása az igényeknek megfelelően dinamikusan növelhető minimális hatással az aktuális, folyamatos működésre
- helyi önállóság: A rendszer tartományokra osztható, ahol az információs rendszer támogatja a helyi szervezeti működéssel kapcsolatos teendőket. A helyi szervezeti működésnek természetesen része az információs rendszer karbantartása, felügyelete és bővítése is.
- nagyobb megbízhatóság és elérhetőség: a központosított rendszerekben egy alkatrész meghibásodása az egész rendszer leállását okozhatja, míg az elosztott rendszerek elemei konfigurálhatók oly módon, hogy meghibásodásuk egymástól függetlenül történjen (redundancia, replikáció). Ilyenkor a meghibásodott alkotóelem helyét egy másik veszi át (teljesítménycsökkenés felléphet), vagy előfordulhat, hogy a hiba folytán egyes felhasználó csoportok elszigetelődnek, de a teljes rendszer, esetleg bizonyos funkciók kiesésével vagy lassulásával, tovább működik.
- nagyobb teljesítmény: A központosított rendszerek működése lelassulhat a kezelt nagy mennyiségű adat és tranzakció hatására. Ha a különböző műveleteket különböző gépek végzik, vagy elhelyezkedésüktől függően egyes felhasználói csoportokat más-más szerverek szolgálnak ki, akkor ez a munkamegosztás a hozzáférés gyorsulásával és válaszidők rövidülésével járhat. Ugyancsak megnövelheti a teljesítményt, hogy a lekérdezések, frissítések végrehajtása a szervezetten belül elosztott adatbázis egyes részein egyszerre történik.
- a biztonsági hiányosságok lokalizáltak: mivel az elosztott rendszerek a biztonság szempontjából különféle elkülönítve kezelt tartományokra oszlanak szét, az egyes adat- és hozzáférés-biztonsági hiányosságok a rendszer többi részére nincsenek hatással

központAz itt leírt előnyök értelemszerűen megegyeznek az elosztott rendszerek hátrányaival.

- egyszerűbb és biztonságosabb adminisztráció: A központosított rendszerek biztonságos adminisztrációja természeténél fogva jóval egyszerűbb (egyetlen biztonsági tartomány, melyet egy helyről kezelnek). Az elosztott rendszerek

esetében bonyolultabb az adminisztráció és a karbantartás megszervezése, összetettebbek a biztonsági és koordinációs eljárások, így maga az adminisztráció is nagyobb költségekkel jár.

- nagyobb megbízhatóság és elérhetőség: Az előző pontban említettük, hogy az elosztott rendszereknél milyen teljesítménynövelő tényezőkre számíthatunk. Számolni kell azonban azzal is, hogy a központosított rendszerek fizikai, környezeti körülményei jobban ellenőrzöttek, és az őket működtető operációs rendszerek mögött évtizedek fejlesztési tapasztalata áll, mely megbízhatóbb rendszert eredményez. Vegyük figyelembe azt is, hogy az elosztott rendszerek sokkal több alkotórészből állnak, így sokkal több a hibalehetőség, ami az elérhetőség csökkenésével jár. Szintén rontják a teljesítményt az egyes komponensek közötti kommunikáció és szinkronizálás által lefoglalt erőforrások.
- képzett fejlesztőgárda és széleskörű felhasználói támogatás: A fejlesztők nagy tapasztalatokkal rendelkeznek, és kiforrott technológiával dolgoznak. A szállított rendszert a szállító pontosan ismeri, így széleskörű, gyors és hatékony felhasználói támogatásra képes. Az elosztott rendszerek esetén a rendszerfejlesztést többen párhuzamosan teljesen elkülönülve végzik, nagyon kevesen látják át a felépülő teljes rendszert. A telepítést gyakran nem is a gyártók, hanem egy erre specializálódott rendszerintegrátor cég végzi. A rendszer különböző komponensei eltérő szállítóktól származnak, így a hibakeresés és azonosítás jóval nehezebb és hosszadalmasabb. A felhasználói támogatás biztosítására sok területet átfogó, magas szintű képzettséggel rendelkező szakembergárdára van szükség.

Elosztott rendszerek működési feltételei

- elkülönítési mechanizmus
- objektum azonosítás
- biztonság

RPC alapelvek: átlátszóság:

- Teljes átlátszóság: Az elosztottsággal behozott komplexitás teljes mértékű lefedése. Általában operációs rendszer szinten kell megvalósítani
- Részleges vagy szelektív átlátszóság: A teljes átlátszóság alkalmazása nem teszi lehetővé, hogy az alkalmazások kihasználják az egyes elemek elhelyezkedésének ismeretéből származó esetleges előnyöket.

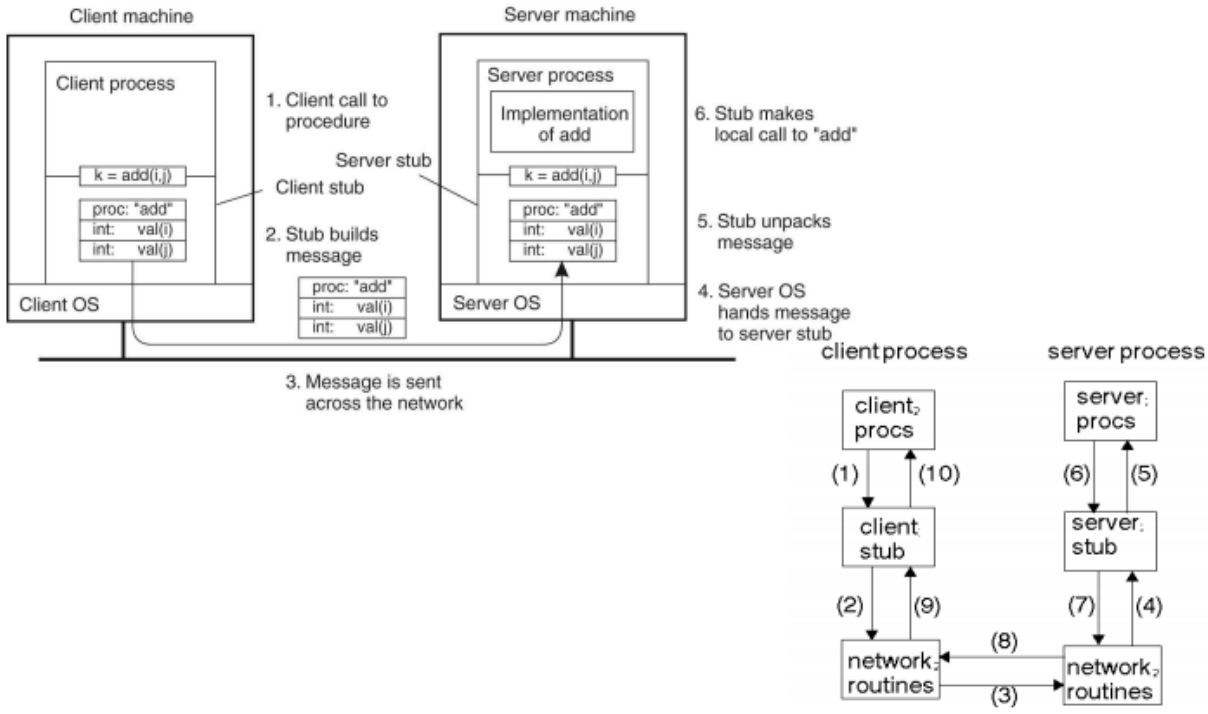
Tíz átlátszó funkció:
• Átlátszó hozzáférés

• Helyfüggetlenségi átlátszóság
• Konkurrens működés átlátszósága
• Átrendezés (migráció) átlátszóság
• Átlátszó replikáció
• Átlátszó partíciónálás
• Állandósági átlátszóság (persistence transparency)
• Hibatűrési átlátszóság
• Teljesítmény átlátszóság
• Skálázási átlátszóság

RPC átlátszóság:

- **Hely-alapú**: A felhasználók / fejlesztők hívásokat indíthatnak, anélkül hogy tudnák, hogy hol van a végrehajtó szerver
- **Funkcionális:** A felhasználók / fejlesztők hívásokat indíthatnak, anélkül hogy tudnák, hogy milyen algoritmusok, illetve optimalizálási eljárások vannak a távoli szerveren.

RPC hívás 10 lépése



RPC Transzparencia

1. Paraméter átadás
   egyetlen adatcsere, értékszerint könnyű, referencia szerint nehéz.
2. Adat reprezentáció (ONC/XDR, DCE/NDR)
   érzékeny a byte sorrendre, XDR, Xerox Courier, NDR
3. Kötés
   tipikusan “name service” vagy “location broker”-hez fordul.
4. Szállítási protokoll
   imlementáció függő: 1 v. több protokoll, UDP, TCP, Shader memory kapcsolat orientált vagy csomagkapcsolt
5. Hibakezelés
   két fajta hiba van: hálózati és szerver. Exceptiönök
6. Hívási szemantika
   exactly-once | at least once | at most once | zero or once
7. Biztonság
   lokális (OS szintű, RPC nem) vagy távoli (egyéni védelem)
8. Teljesítmény
   Lassulhat: protokoll, kontextus váltás, adatmásolás hálózat miatt

Com jellemzők (10)

1. Dinamikus linkelés
2. Szeperáljuk az interfészt és implementációt külön osztályokba
3. Az interfészben nem szerepeltethetünk változókat
4. Az interfész metódusai „pure virtual”
5. Egy interfész egyszerre csak egy interfészből származhat, de akár hányszor
6. Az interfész kiterjeszthető
7. Egy implementáció több interfészt implementálhat
8. A komponens működéséhez szükséges műveletek egy generikus interfészbe kerülnek, ebből származik minden interfész (root interfész)
9. A COM root interfésze IUnknown(QueryInterface, Addref, Release)
10. Az interfész metódusok visszatérési értéke HRESULT
11. Minden Interfészbeli metódusokban, minden paraméternek van attribútuma
12. A COM hívásai alapvetően szinkron típusúak
13. COM speciális interfészeket kínál(IClassFactory, IConnectionPoint, Idispatch)

esetleg még: szálmodell, apartmanok,

Middleware szolgaltatasok

Olyan szolgáltatások, melyeket általában a középső réteg

valósít meg
• Távoli eljáráshívás
• Szálkezelés
• Terheléskiegyenlítés
• Átlátszó hibakezelés
• Perzisztencia
• Tranzakciókezelés
• Objektumok életciklusa
• Aszinkron üzenetkezelés
• Biztonság

Middleware típusok
• Explicit
> Alapvetően API hívások révén
> Tipikusan a CORBA használja
- felduzzad a forráskód, nem rugalmas

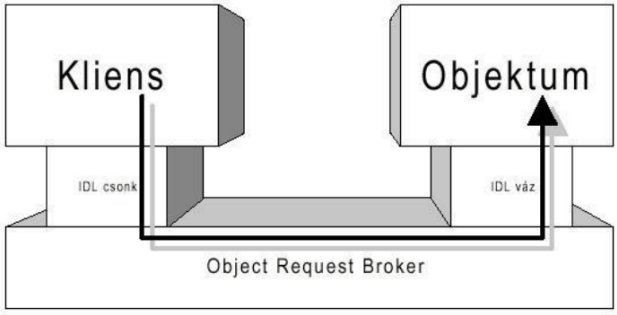
• Implicit vagy deklaratív
> Attribútumok(COM+)
> XML Descriptor (EJB)
- külön leíró fájl, hogy mely szolgáltatások kellenek
- forráskód csak üzleti logika
- könnyű változtatás

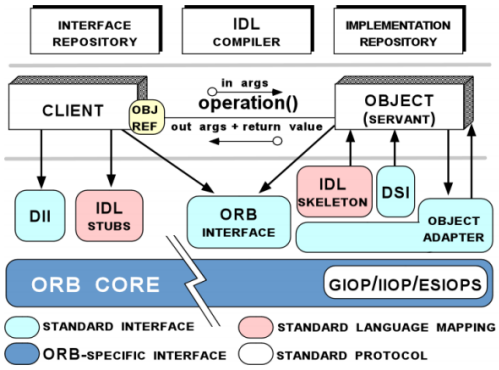
• Természetesen az implicit middleware a preferált típus. Szétválik a fejlesztés és az addminisztráció (Component explorer)

2.4 aktivalasi módok

• Client Actiovation (CAO – Client Activated Object, Activated): A kliens kérésére jön létre az objektum a szerver oldalon.
• Server Activation (SAO – Server Activated Object, Well known): A szerver kontrolálja az objektumok létrehozását. Két esete van:
> SingleCall: minden kliens kéréshez létrejön egy új objektum szerver oldalon. A metódus meghívása után nincsen már rá szükség, a GC eltávolítja.
> Singleton: a szerveren egy példány jön létre az objektumból, és ez szolgál ki minden klienst.

Corba alap architektúra





Corba services (10)

- Lifecycle Service,
- Relationship Service,
- Naming Service,
- Persistent Object Service,
- Externalization Service,
- Event Service,
- Object Query Service,
- Object Properties Service,
- Object Transaction Service,
- Concurrency Service,
- Licensing Service,
- Trader Service,
- Security Service,
- Time Service,
- Collection Service,
- Notification Service

Any - variant

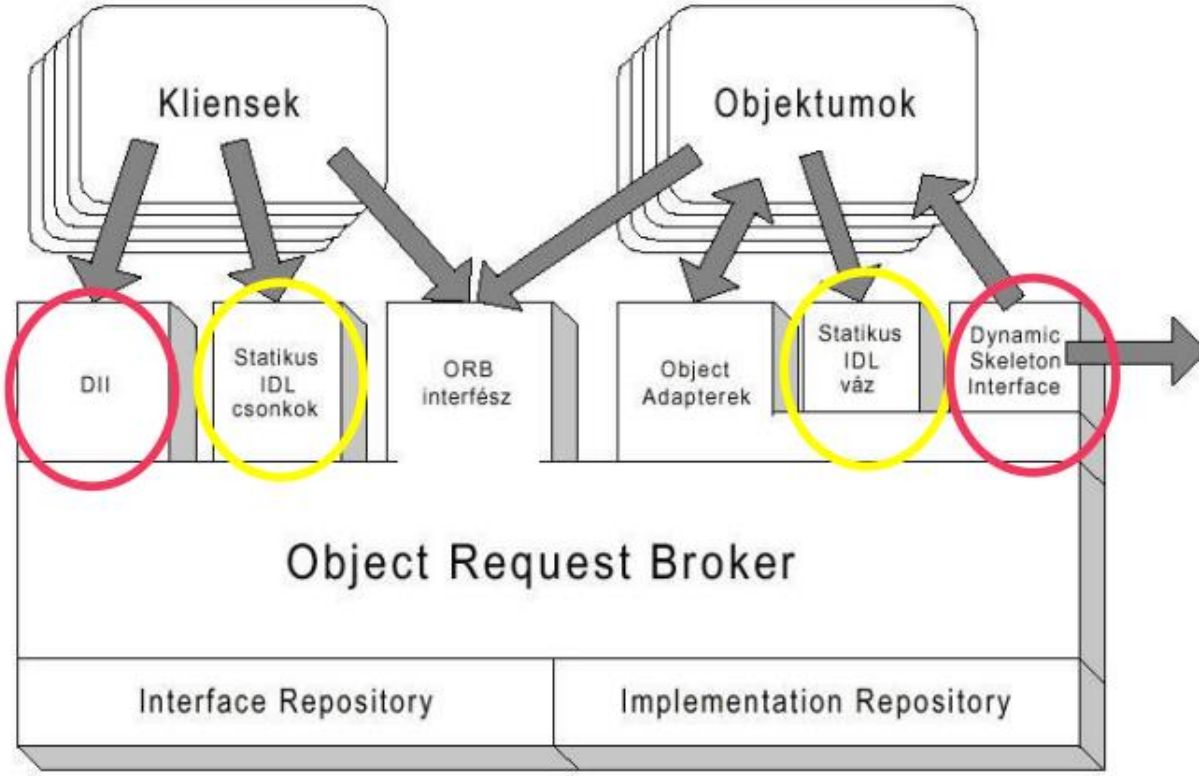
passz

Idl interfész com, Corba

- Minden interfész metódus virtual public
- Nincsenek objektumváltozók
- A struktúrában csak adatok vannak
- Nincsen konstruktor, destruktor
- A régi CORBA-ban az interfészek között volt többszörös öröklés, de ezt később kivették. Az új verziókban az implementáció több interfészt

implementál, a régiben egyet támogattak

Az orb kliens/objektum felőli része



Az objektum adapterek

• Implementációk regisztrálása,
• Objektumreferenciák létrehozása és generálása,
• Objektumreferenciák leképzése az hozzájuk tartozó implementációkra,
• Implementációk aktiválása és deaktiválása,
• Kérések továbbítása a vázon vagy a DSI-on keresztül,
• A biztonság szavatolása a Security Service segítségével.
• **Két alap típusa**

- **BOA**, - Basic Object Adapter (négy féle kapcsolat a szerver, és objektum között: Osztott, Állandó, Egyedi, funkciónkétni szerver)
- **POA** - Portable Object Adatpter (összetettebb mint a BOA, általános célokra, hierarchikus)

Szereplők a poa világában

• Kliens: olyan kódrészlet, amely egy objektumon valamilyen műveletet végrehajt.
• Implementáció: az IDL interfészben definiált objektumot megvalósító kód.
• Szerver: olyan kód, amely egy vagy több IDL interfész implementációját tartalmazza.
• Servant: Egy interfész implementációjának a példánya
• Object ID: egy POÁ-ban lokálisan értelmezett az objektum referenciától független objektumazonosító, amellyel a POA és az implementáció az absztrakt CORBA objektumot azonosítják. A POA ennek segítségével keresi ki az általa ismert Servant-ok közül a megfelelõt. Ez az azonosító a kliensek számára rejtett. Az Object ID-t vagy a szerver vagy a POA határozza meg.
• Object Reference: tartalmazza a POÁ-ra való hivatkozásokat és az Object ID-t valamilyen formában
• Active Object Map: Egy adott POÁ-hoz tartozó Active Servant-ok azonosítóinak tárolására szolgál
• Active Servant: olyan Servant, amely a POA aktív objektum térképén (Active Object Map) rajta van
• Root POA: az ORB által alaphelyzetben biztosított POA. A POA fa első eleme
• POA Manager: POA objektumok aktiválására és deaktiválására használatos
• Policy: a POA viselkedését meghatározó attribútum, meghatározza például a kéréstovábbítás, objektum-karbantartás, illetve többszálas futás kezelésének módját
• Servant Manager: az implementáció által biztosított opcionális funkcionalitás, amely aktivál vagy deaktivál egy adott Servant-ot, az Object ID alapján.
• Adapter Activator: a szerver által biztosított objektum, amelyet az ORB akkor szólít meg, amikor egy adott objektumreferenciához tartozó POA nem létezik

Giop részei, üzenetek

- General Inter ORB Prokoll (egyszerű, méretezhető, architekturálisan független)
- • Az általános adatábrázolási definíció (Common DataRepresentation - CDR)
  > Byte sorrendiség, Adatillesztés
- • A GIOP üzenetek formátuma
  > 7 üzenet, 3-kliens-szerver, 3 szerver kliens és 1 kétirányú
- • A GIOP szállítási feltételek
  > Kapcsolat alapú protokoll
  > Megbízható adattovábbítás
  > A kapcsolat felépítése hasonlóan kell lezajlania, mint a TCP/IP protokollnál.
- **Üzenetek típusai:**

- RequestMessage: kérést továbbító üözenet
- ReplyMessage: reakció rá
- Cancel Request: végrehajtás törlése
- LocateRequest: érvényes-e az objektumreferencia, fel tudja-e dolgozni a szerver?
- LocateReply: LocateRequest válasza
- CloseConnecton: kérem kaccsojjaki
- MessageError: hibás volt az üzenet

Logo példa dia feladata más nyelvvel + elmélet

lásd: mintaZH kidolgozás

Szakteruleti nyelv fogalma

Szakterületi nyelv (Domain-Specific Language,DSL)
Programozási, vagy specifikációs nyelv korlátozott kifejezőerővel, egy konkrét szakterület problémáinak és megoldásainak leírásához.

Internal, external DSL

A nyelv jellege szerint
> Internal DSL
– Általános célú programozási nyelv speciális módon használva
– A nyelvelemek közül csak néhányat használunk
– Pl. script nyelvek; saját framework hívások
> External DSL
– Saját nyelv, nem az alkalmazás programozási nyelve
– Saját szintaxis, vagy egy más nyelv szintaxisa
– Pl. a PicProcessor nyelv; Unix parancsok; SQL

Összefoglaló diák

well…

Miből áll a vizuális Szakteruleti nyelv?

• Mire van szükség egy vizuális szakterületi nyelv definiálásakor?
> Nyelv struktúrája
> Kiegészítő kényszerek - Absztrakt szintaxis
> Megjelenítés -Konkrét szintaxis

> Struktúra jelentése -Szemantika

Ocl kifejezés típusok

- **Kontextus:** modell elem, amre értjük a megkötést (context Customer)
- **Invariáns** (inv name = “Gandalf”)
- **Elő- és utófeltétel** (pre valami: logika) (post valami: logika)
- **Kezdeti értékek** (init: set())
- **Származtatott érték (**derive: …)

Invarians gyak példa

passz
diából:
context Customer
inv: self.name = ’Edward’

context Customer
inv: age >= 18

context CustomerCard
inv checkDates: validFrom.isBefore(goodThru)

Művelet előtti értékre lehet apellalni

Amikor preconditiont adsz meg, akkor a @Pre azt jelenti a kifejezésben, hogy a művelet előtti érték.

Customercard kisnagy betű

wat - Navigációnál merül fel példa a Diasorban.

Feladat: szemantikai vagy Ocl + ebnf biztos

lásd: ZH
lásd: OCL_példák.pdf a tárgyoldalon
pl.:• Pacmannek három élete lehet

```
context Pacman:
inv: lives <= 3
```

• Csak pacman és a szellemek tudnak mozogni

```
context Figure:: canMove()
pre: isOclTypeOf(Pacman) or isOclTypeOf(Ghost)
```

• Mozgási szabály

context Figure:: move(f: Field)
pre: canMove()
post: self.position = f

• Ha a szellem olyan mezőre lép, ahol nem szuper pacman van, a pacman élete eggyel csökken.

```
context Ghost:: move(f: Field)
post:
    let: pacmans : Set{ Pacman} = self.field.figures -> select(oclIsTypeOf(Pacman))
in
    if pacmans->size() = 1
    then pacman.lives = pacman.lives@pre – 1
    endif
```

.

EBNF - LOGO

- A LOGO programok parancsokból (**Command**) állnak.
- Három fajta parancsot ismerünk: **Pen**, **Move** és **Repeat**. A parancsok végén mindig van egy szóköz, vagy egy új sor karakter (tagolásként)
- A **Pen** parancs két fajta lehet: **PENUP**, vagy **PENDOWN**. (A leírásnak nem része, de a LOGO nyelv szemantikáját is ismerve az előbbi utasítás a rajzolás felfüggesztésére, míg utóbbi a rajzolás elkezdésére ad utasítást.)
- A **Move** parancs három fajta irányt fogad el (**FWD/LEFT/RIGHT**) és egy számot. (Szemantika ismeretében: a mozgás irányát és mértékét adjuk itt meg.)
- A **Repeat** parancs egy számot igényel, majd utána szögletes zárójelek közt felsorolhatunk parancsokat. (Szemantika ismeretében: ismételni fogja a megadott parancsokat a megadott számszor.)
- Végül a **Num** és a **Digit** szimbólumok definíciójára van szükség.

<<nyomtatáshoz bemásoltam ide a ZH-t is >>
Elosztott rendszerek minta ZH

1. Hasonlítsa össze az elosztott es központosított rendszereket! (10p)

Elosztott	**Központosított**
Jól skálázható	egyszerűbb és biztonságosabb adminisztráció
Nagy rendelkezésre állású	képzett gárda
Lokálisan önálló	Megbízhatóság és elérhetőség
Terhelés elosztás	
Nagyobb teljesítmény	
Jobb ár / érték arány	
Rugalmasság	

2. Soroljon fel 10 middleware szolgáltatást es fejtsen ki ötöt részletesen! (15p)
Távoli eljáráshívás
Szálkezelés
Tranzakciókezelés
Névfeloldás
Biztonság
Objektumok életciklusának kezelése
Terhelés elosztás
Aszinkron üzenetkezelés
Perzisztencia
Hibakezelés

3. Mi a különbség az implicit es explicit middleware között? (10p)
Implicit:

- Szétválik a fejlesztés és az adminisztráció
- Attribútumok(COM+)
- XML Descriptor (EJB)
- Külön leíró fájl tartalmazza, milyen middleware szolgáltatásokat veszünk igénybe
- A Kérésmegszakító a leíró fájl alapján generálódik
- A forráskód valóban csak üzleti logikát tartalmaz
- A leíró fájlt módosíthatja a vevő, a forráskódot nem kell kiadnunk

- Egy lehetséges megvalósítás az interfész és implementáció szétválasztása

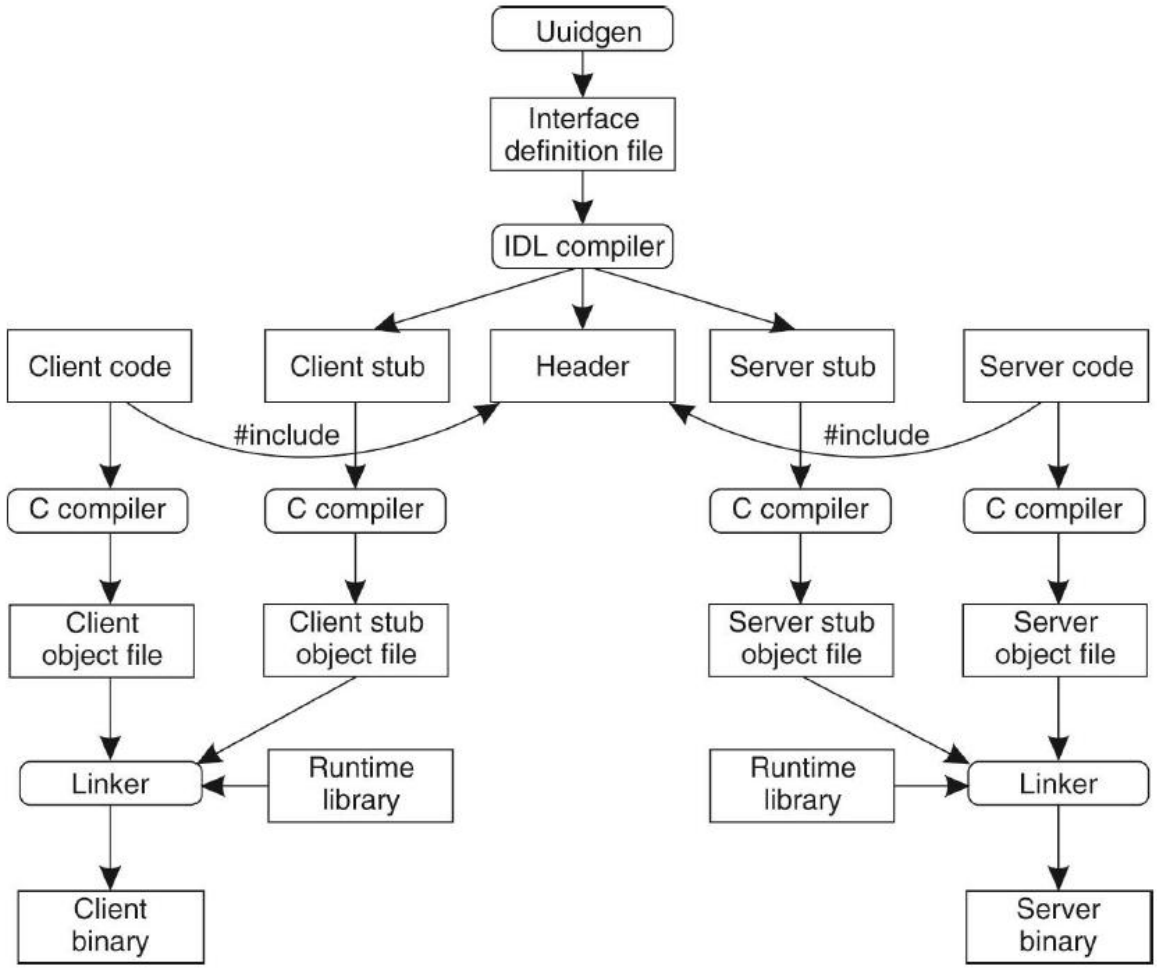
Explicit:

- Alapvetően API hívások révén
- Tipikusan a CORBA használja
- Felduzzad a forráskód
- Nem rugalmas a middleware (ha eladjuk a komponenst, ki kell adni a forráskódot, ha a vevő pl. más tranzakciókezelést akar)

4. Soroljon fel 5 különbséget a COM es CORBA között! (10p)

- COM csak Windows-on, CORBA platform független
- COM-ban nincs tranzakció kezelés, CORBA-ban van
- CORBA-ban kifinomultabb biztonsági modell mint COM-ban
- COM implicit middleware, CORBA explicit middleware
- COM root interfész: IUNKOWN, CORBA root: CORBAObject
- COM-ban egyszeres öröklés, másikban többszörös
- COM-ban az attribútumok nem támogatottak, CORBA-ban get;set;-re fordul

5. Vázolja fel az RPC alapú fejlesztési folyamatot! (15p)



6. Mi az absztrakt szintaxis és mi a konkrét szintaxis a vizuális nyelveknél? (4p)

Konkrét szintaxis: a szakterületi függő megjelenítés definíciója
Absztrakt szintaxis: a nyelv absztrakt jelentésének definíciója (amit a metamodellel adunk meg általában)

7. Mit nevezünk external ill. mit nevezünk internal szöveges szakterületi nyelvnek? (4p)

External: Külön specifikációs nyelvet használunk pl PictureProcessor nyelv
Internal: Egy programozási nyelvet használunk erre a célra, de nem az összes szimbólumát. pl script nyelvek

8. Értékelje az alábbi állítást: A OCL egy gyengén típusos, imperatív nyelv, ami a MOF metamodellekben található és az elemek közti kapcsolatok megkötéseinek szabályait hivatott leírni. (7p)

- Nem gyengén hanem erősen típusos
- Nem imperatív, hanem deklaratív
- Nem csak elemek közötti kapcsolatokat, hanem metamodell elemek belső működésére/állapotára is tehetők megkötések
- MOF metamodelleken használható, de hogy bennük található nem tudom mennyire helytálló

9. Mik a denotációs szemantika alapelvei, módszerei? Szemléltesse őket röviden (!) a hexadecimális számok körében elvégzett négy alapművelet szemantikai leírásával! (10p)

- A művelet hatására koncentrál
- Három részből áll:
    - Nyelv (a cél szakterület)
    - Szemantikai algebra (számítási modell leírására)
    - Leképző függvény

10. Definiálja a következő nyelvet EBNF segítségével: bábuk vezérlése a sakktáblán. A bábuknak a színét, típusát és egyedi azonosítóját megadva meghatározhatjuk, hogy hova lépjen. Legyen lehetőség elágazások megadására az ellenfél bábuira való feltételek megadásával (pl. ha egy gyalog áll a B3-on, akkor lépjünk a C3-ra, különben a D4-re)! Csak érvényes mezőkód fogadható el, de az ütést nem kell kezelni a nyelvtanban. Opcionálisan lehessen megadni egy új bábu típust is a lépésnél, hogy modellezhessük, ha a gyalog beér az utolsó mezőre és átváltjuk egy másik bábura. Ne csak egy lépést lehessen leírni egy scripttel, hanem egy egész játszmát! (15p)

//Alap struktúrák definiálása
Számjegy = 0|1|2|3|4|5|6|7|8|9;
Szám = Számjegy – “0”, {Számjegy};

//Játék specifikus struktúrák
Azonosító = „ID_”, Szám;
Mező = A|B|C|D|E|F|G|H , 1|2|3|4|5|6|7|8 ;
Típus = “gyalog”|”bástya”|”huszár”|”futó”|”király”|”vezér” ;
Szín = „világos”|”sötét”;
Bábu = (Azonosító, ’’,’’ , Szín, ’’,’’ ,Típus)

Játszma = {Parancs};
Parancs = (HA | LÉP);

HA = “HA”, Feltétel, “AKKOR:”, Parancs, “KÜLÖNBEN:”, Parancs;
Feltétel = “BÁBU_ÁLL”, (Mező, ’’,’’ , Típus);

LÉP = Bábu, „LÉPJEN”, Mező, [Típus]

//Példa játék, szicíliai védelem
(ID_5, világos, gyalog) LÉPJEN E4
(ID_12, sötét, gyalog) LÉPJEN C5

//Feladat kiírás feltétel vizsgálata
HA BÁBU_ÁLL (B3, Gyalog) AKKOR:
(ID_7, világos, vezér) LÉPJEN C3
KÜLÖNBEN:
(ID_7, világos, vezér) LÉPJEN D4

//Átalakulás példa
(ID_11, sötét, gyalog) LÉPJEN A1, Vezér

Wiki-ről copy-paste:

Aki már valameliyk kérdésre a választ kiírta, kijegyzetelte, vagy megtanulta, leírhatná.

~~Idén szerintem nem voltak~~

Tipikus ZH/Vizsga kérdések

- **elosztott rendszerek előnyei a központosított rendszer előnyeivel**
- **GIOP protokoll (General Inter ORB Protocol) üzenet típusai, üzenet tartalma**
- **COM objektum típusok**

2013.06.06 vizsga

1. **GIOP ismertetése (15 pont)**
2. **10 db Middleware szolgáltatás, ebből 5-öt részletesen kifejteni (15 pont)**
3. **COM és CORBA technológiák különbségei (10 db különbség) (20 pont)**
4. ~~**Objektum relációs leképezés fogalmai, hogyan oldható ez meg JPA-val. (15 pont)**~~
5. **EJB-ben időzítés megoldása+szekvencia diagram (15 pont)**
6. ~~**Milyen problémát old meg az XML web szolgáltatások?, mi a megoldás kulcsa?. Mik a hozzá kapcsoló szabványok? Mik a WS-* szabványok?, sorolj fel hármat. (20 pont)**~~

2013.05.30 vizsga

1. **elosztott rendszerek vs centralizált különbségei**
2. **COM interfészek felsorolása (5db), részletezd**
3. **GIOP**
4. **integrációs megoldásokból 4 db**
5. **EJB tranzakciós attributumok, mire jók, sorold fel, részletezd**
6. **.NET remoting fogalmai, működése általánosságban, hogyan lehet objektumokat létrehozni**
7. **WCF: mi és mire jó a binding, objektumok szálkezelése**

- **ElosztottRendszerekVizsga20050526**
- **ElosztottRendszerekVizsga20110612**

ZH

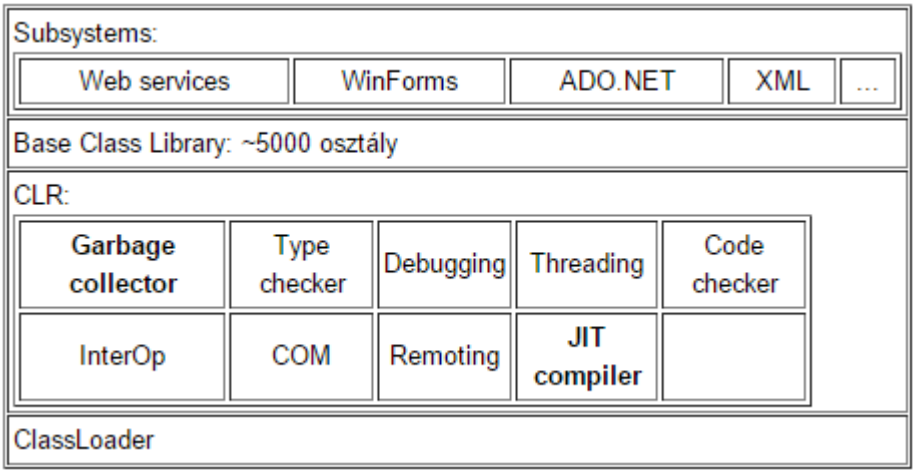
2006.04.24. minta zh

1. **Kifejteni miért fontos az elosztott rendszer (centralizált/elosztott rendszer összehasonlítása).**
    - **centralizált rendszer előnyei**
        - **könnyen adminisztrálható**
        - **nagy megbízhatóság redundáns hardverrel biztosítható**

        - szakértőket biztosít a szállító
    - elosztott rendszer előnyei
        - rugalmas
        - horizontálisan is skálázható
        - nagy teljesítményű
        - dinamikus feladatelosztással megbízhatóvá tehető
        - jó ár/teljesítmény
        - a rendszer bizonságkritikus részei jól szeparálhatók
2. Komponens alapú fejlesztés előnyei és hátrányai.
    - komponensek külön fejleszthetők
    - interfész és implementáció külön van választva
    - interfész is bővíthető (örökléssel vagy aggregációval)
    - elég csak a bináris kódot kiadni a megrendelőnek
    - konténer biztosítja a middleware-t szabványos felületen keresztül
    - deklaratív leíró file, adminsztrációs felület biztosított hozzá
    - komponens technológiák egymás között nem átjárhatók
3. Milyen típusú servereket ismer a COM-ban?
    - in-process: komponens a kliens processzében fut. Gyors, de csak szinkron hívás van és egy hibás komponens magával ránthatja a klienst is. Pl. VB
        - in-process handler: felüldefiniálható a standard marshalling. Pl. .NET Application Domains
    - local server (out-process): a szerver (tipikusan .dll) külön processzben fut, ha elszáll, a kliens csak timeoutot kap. Stabil, de lassabb, mint az in-process
    - remote server: a szerver távoli gépen is futhat, a hozzáférés transzparens. Ez jelenti a legnagyobb overheadet. Pl. DCOM
4. Middleware szolgáltatások (10 db), ezek közül néhányat kifejteni.
    - névfeloldás, security, tranzakciókezelés, object pooling, perzisztencia, load balancing, életciklus management, szálkezelés, event/notify, messaging
5. GIOP protokoll.
    - GIOP Fejléc: magic string, verzió, byte sorrend, üzenet típus (1-7), üzenet méret
        - RequestMessage (K->S) — kérés: GIOP header, Message header (objektum azonosító, metódus, szolgáltatások, aszinkron kérés azonosító), Body (metódus paraméterek)

- **ReplyMessage (S->K) — válasz a kérésre: GIOP header, Reply header (válasz azonosító (mire válasz?), státusz kód), Body (visszatérési érték, hibainfó)**
- **CancelRequest (K->S) — aszinkron kérés megszakítása: GIOP header, kérés ID**
- **LocateRequest (K->S) — objektum megpingelése: GIOP header, objektum ID**
- **LocateReply (S->K) — ping válasz**
- **CloseConnection (S->K) — kapcsolat befejezése**
- **MessageError (K<->S) — hiba**

6. **~~.NET framework fő részei (esetleg volt szó .NET remotingról, de erre pontosan nem emlékszem).~~**



Bővebb infó angolul itt

7. **~~Web service, milyen célra használható?~~**
   - **~~integráció különböző platformok között~~**
   - **~~külső cég által fejlesztett komponensek felhasználása~~**
   - **~~üzleti folyamatok tervezése~~**
   - **~~fejlesztési paradigma~~**