Elosztott rendszerek ZH kivonat.

2015

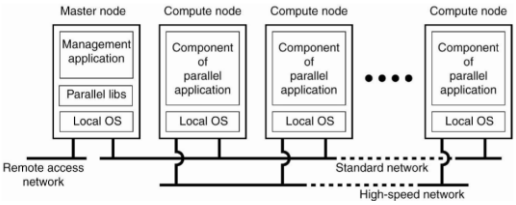
Ez a kivonat saját használatra készült, lehetséges, hogy túl pongyolán fogalmaz, illetve kihagy dolgokat. Nem hivatalos, hivatkozási alap nem lehet. Forrás: 2015 tavaszi előadás diák. Megfelelő körültekintéssel kezeld.

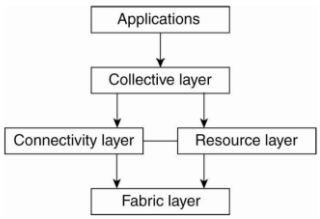
Készítette: Lukács Ferenc, feco911@gmail.com

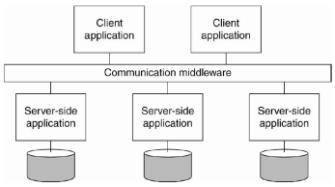
1.Ea

- Elosztott rendszernek nevezzük az **autonóm**, egymással hálózaton **együttműködő** számítógépeket, amelyek közös együttműködésén alapuló szolgáltatását a felhasználó úgy érzékeli, mintha azt egy **integrált** eszköz szolgáltatta volna.
- Történelem
  - 60-as évek ~ **mainframe**: könyvelési automatizálási megoldások, hatékony tranzakció feldolgozás
  - 70-es évek ~ **miniszámítógépek**: egyes vállalati osztályainak külön-külön gép
  - 80-as évek ~ **személyi számítógépek**: könnyű használat
  - 90-es évek: HW és SW együttes tovább fejlődése
    - elosztott rendszerek előtérbe kerülnek: egy központi DB és sok kliens PC
  - 2000-es évek: szolgáltatás alapú megközelítés, több fajta eszköz támogatása
- SW fejlesztés során 4 részterület
  - szervezet és ügyfelei összekapcsolása
  - szervezet belső eljárásainak integrációja
  - információn alapuló szolgáltatások és termékek
  - döntéshozó rendszerek
- Elosztott rendszerekkel szembeni elvárások
  - ügyfelek közvetlen elérése
  - szolgáltatások minőségének javulása
  - gyors, egyszerű információ frissítés
  - meglévő infrastruktúra jobb kihasználása
- Fejlesztési paradigmák evolúciója
  - Monolit -> OO -> Komponens orientált -> Szolgáltatás orientált -> Szolgáltatások és kütyük
- Elosztott formák
  - Klaszter: van egy master, és több slave. A master dönti el, hogy melyik slave végezze el a munkát, mindezt a felh számára transparensen
  - Grid
  - Tranzakciós feldolgozó rendszerek: Kliens tranzakciókon keresztül kommunikál a TP monitorral, az pedig szétossza a feladatot több szerver között.
  - P2P: Számítógépek egymáshoz kapcsolódhatnak egy tracker segítségével. A trackertől eltekintve a számítógépek egyenrangúak. Egyszerre kliens és szerver is egyben minden gép.
  - Enterprise rendszerek:Kliens alkalmazások egy kommunikációs middlewaren keresztül kommunikálnak a szerver oldali alkalmazásokkal
  - SOA
  - Sensor hálózatok

Klaszter Grid SOA







- Elosztott vs központosított rendszerek

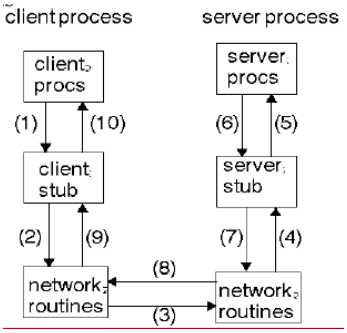
Elosztott	**Központosított**
Rugalmasság	Egyszerűbb és biztonságosabb adminisztráció
Skálázhatóság	Megbízhatóság és elérhetőség
Lokálisan önálló	Képzett gárda
Jobb megbízhatóság és elérhetőség	
Nagyobb teljesítmény	
Biztonsági megoldások lokalizálhatósága	

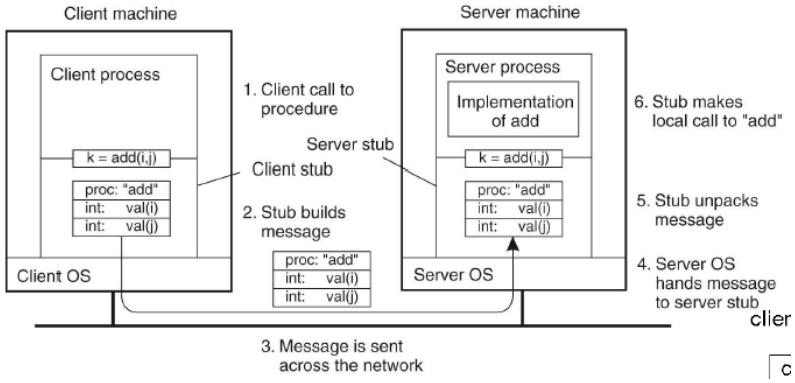
- Átlátszó elosztottság
  - Teljes átlátszóság: Komplexitás teljes mértékű lefedése. Általában OS szinten kell megvalósítani
    - alkalmazásával nő a komplexitás
      - Átlátszó hozzáférés
      - Helyfüggetlenségi átlátszóság
      - Konkurens működés átlátszósága
      - Átrendezés (migráció) átlátszóság
      - Átlátszó replikáció
      - Átlátszó partíciónálás
      - Állandósági átlátszóság (persistence transparency)
      - Hibatűrési átlátszóság
      - Teljesítmény átlátszóság
    - Skálázási átlátszóság
  - Részleges / Szelektív átlátszóság: A teljes átlátszóság nem teszi lehetővé, hogy az alkalmazások kihasználják az egyes elemek elhelyezkedésének ismeretéből származó előnyöket
- Elosztott struktúrák
  - Mester-szolga
  - Kliens-szerver
  - Egyenrangú (P2P)
  - Csoport
  - Elosztott objektum
  - Multimédia áramlás

2.Ea

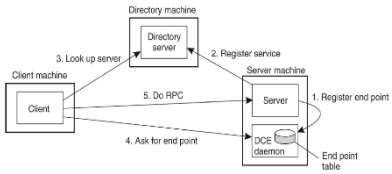
- Kommunikáció távoli eljárások között
  - Eljárás regisztráció
  - Kommunikációs csatorna kiépítése az eljárások között
  - Megbízható üzenettovábbítás
  - Szinkronizáció a folyamatok között
  - Kommunikációs csatorna lezárása.
- Kötés (Binding)
  - Statikus: fordításidőben, egyszerű, gyors, rugalmatlan
  - Dinamikus: futásidőben, lassabb de rugalmas, könyvtár szolgáltatás szükséges

    - Explicit
- Átlátszóság
    - Hely alapú: A felhasználók / fejlesztők hívásokat indíthatnak anélkül hogy tudnák, hogy hol van a végrehajtó rendszer.
    - Funkcionális: A felhasználók / fejlesztők hívásokat indíthatnak anélkül, hogy tudnák, hogy milyen algoritmusok illetve optimalizálási eljárások vannak a távoli rendszeren.
- RPC
    - Régi, de fejlődik
    - Platform független, heterogén környezet támogatása
    - Szorosan csatolt kliens-szerver architektúrát követ.
    - Default szinkron, de lehet aszinkron is.
    - Nem kell tudni a hívott eljárás helyét és algoritmusát.
    - IDL (Interface Definition Language)-et használ.
- RPC hívások 10 lépése
    - Kliens oldali eljárás hívás
    - Kliens oldali stub létrehozza az üzenetet
    - Üzenet átküldése a hálózaton
    - Szerver oldali OS fogadja az üzenetet továbbítja a szerver oldali stubnak
    - Szerver oldali stub kibontja az üzenetet
    - Szerver oldali stub lokálisan meghívja a kért függvényt
    - Ugyanígy visszafele.





- Kötés létrehozásának lépései
    - Szerver regisztrál egy end pointot
    - Szerver beregisztrálja a szolgáltatását a könyvtár szolgáltatásba
    - Kliens megnézi a könyvtár szolgáltatást
    - Kliens lekérdezi a szervert end point-ját
    - Kliens RPC-t tesz a szerver end point-ja vele



- Socket-ek
    - Socket: Új end point létrehozása
    - Bind: sockethez lokális cím rendelése
    - Listen: adott porton való figyelés a bejövő kapcsolatok miatt
    - Accept: elfogadja a kapcsolódást
    - Connect: kapcsolat létrehozása
    - Send: adat küldés a kapcsolaton keresztül
    - Receive adat fogadás a kapcsolaton keresztül
    - Close: kapcsolat lezárása
- RPC motivációk

    - Elosztott rendszerek fejlesztésének egyszerűsítése
    - Automatikusan kezelik
        - Megbízhatóság (tranzakció)
        - Platform heterogenitás
        - Szolgáltatás hely kiválasztás
        - Szolgáltatás aktiválás és megfigyelés
        - Biztonság
- Problémák
    - Kliensről szervernek küldött kérés elveszik
    - Szerverről kliensnek küldött válasz elveszik
    - Szerver crash miután vette a kérést.
    - Kliens crash miután elküldte a kérést.
- Esetleges megoldások
    - Exactly-once: pontosan egyszer érkezik meg a kérés, gyakorlatban lehetetlen
    - At least once: legalább egyszer megérkezik a kérés
    - At most once: legfeljebb egyszer(0, nemtudom,1)
    - Zero or once: egyszer vagy egyszer sem: tranzakciós
- RPC transzparencia
    - Paraméter átadás
        - egyetlen adatcsere az RPC-ben
        - érték szerint: egyszerű; másolat képződik
        - referencia szerint: nehezebb
    - Adat reprezentáció
    - Kötés
        - Kliens egy naming service-hoz fordul
    - Szállítási protokoll
    - Hibakezelés
        - Lehet hálózati illetve szerver hiba
        - Tipikusan Exception-ök
    - Hívási szemantika: ld esetleges megoldások
    - Biztonság
        - Lokális: folyamatok OS szintű biztonsága
        - Távoli
    - Teljesítmény
        - Lassulás lehet protokoll, kontextus váltás, adatmásolás, vagy hálózati okok miatt

3.Ea

- Statikus linkelés a memóriában: többször van a memóriában ugyanaz az adat, a módosítása is problémás.
    - Dll: megoldja a statikus linkelés problémáját
    - egyszer kell tárolni csak a közös kódot
    - de nem elég csak a tagfüggvényeket exportálni a DLL-ből az újrafelhasználhatósághoz, mert a C++ megköveteli, hogy a fordító férjen hozzá az osztály információihoz beleértve a private és a protected adattagokat is.
    - Megoldás: interfész és implementáció szétválasztása külön osztályba
        - interfészben nem lehet olyan változó, amit használ az implementáció
        - interfészben csak tiszta virtuális függvények lehetnek
        - interfész egyszerre csak egy interfésztől származhat

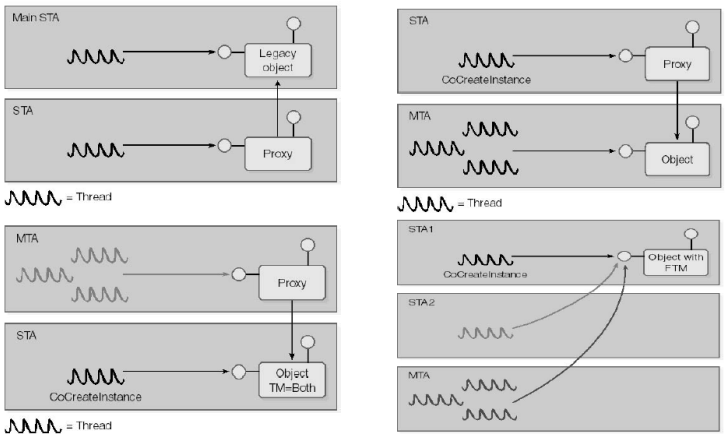
        - egy implementáció több interfészt is implementálhat
        - ha megváltozik az implementáció, attól még a klienst nem kell megváltoztatni, mert az interfész ugyanaz maradt

4.Ea

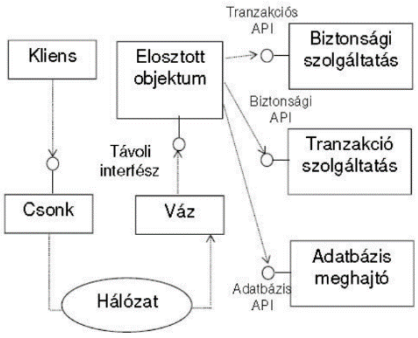
- COM ~ Component Object Model
    - Interfészek: IClassFactory, IConnectionPoint,IDispatch
    - Jellemzők
        1. Dinamikus linkelés
        2. Szeperáljuk az interfészt és implementációt külön osztályokba
        3. Az interfészben nem szerepeltethetünk változókat
        4. Az interfész metódusai „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)
    -
    - Szál szinkronizációs módszerek: Semaphore, event,mutex,critical section
    - Apartman típusok
        - a COM objektumokhoz való hozzáférés szálkezelési szabályait rögzíti
        - Típusai
            - STA – Single-threaded Apartment ~ egyszálú apartman
                - egy objektum abban az STA-ban jön létre, amelyikek a szálából példányosítottuk. Az objektum metódusait mindig ugyanaz a szál fogja végrehajtani, amelyik létrehozta az objektumot. Így lehet tárolni adatokat lokális helyen, nem lesz versenyhelyzet sehol sem.
                - COINIT_APARTMENTTHREADED
                    - automatikusan létrejön egy message várakozási sor is az STA-hoz
            - MTA – Multithreaded Apartment ~ többszáló apartman
                - egy szál jelzi, hogy MTA-ba be szeretne lépni: ColninitializeEx COINIT_MULTITHREADED paraméterrel történő meghívásával
                -
            - NA – Neutral Apartment ~ semleges apartman
        - Minden folyamathoz több apartman tartozhat, de csak egy MTA és egy NA lehet, míg több STA is.
        - MTA és NA-ban több szál is futhat, szinkronizáció megoldása a mi feladatunk

- STA általában felhasználó által vezérelt graf felülettel rendelkező objektumoknál használatos,
- MTA a nem látható, feladatot végző komponenseknél (worker) használatos

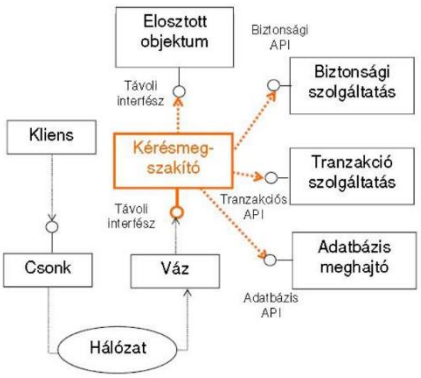
A 3 szálmodel együttműködése



- COM hiányosságok
  - Csak Windows
  - Alapvetően szinkron,
  - Apartman kiegészítés, bonyolult
  - Nincs tranzakciókezelés
  - Rövid élettartamú objektumok létrehozása időigényes
  - Túl kevés vagy túl sok szál – nem hatékony
  - Nem elég kifinomult a biztonsági modell
  - Lazán csatolt rendszerekhez túl szigorú
  - Nem elég rugalmas eseménykezelés
- Middleware szolgáltatások
  - olyan szolgáltatások, amelyeket általában egy középső réteg valósít meg
    - távoli eljáráshívás
    - szálkezelés
    - terhelés elosztás
    - átlátszó hibakezelés
    - perzisztencia
    - tranzakciókezelés
    - objektumok életciklusa
    - aszinkron üzenetkezelés
    - biztonság
  - típusok
    - explicit: API hívások révén, CORBA használja
      - hátrányai
        - felduzzad a forráskód
        - nem rugalmas: komponens eladásával el kell adnunk a forráskódot is
    - implicit: attribútumok (COM+), vagy XML leíró (EJB)



        - külön leíró fájl tartalmazza milyen middleware szolgáltatásokat veszünk igénybe
        - kérésmegszakító a leíró fájl alapján generálódik
        - forráskód valóban csak üzleti logikát tartalmaz
        - forráskódot nem kell kiadni
        -
    - implicit típusú az ajánlott.



- MTS ~ Microsoft Transaction Server
    - COM komponens
    - Szál, biztonság, tranzakció felelőse
- COM+ middleware szolgáltatások
    - szálkezelés, application pooling
        - segítségével egyszálú komponensek skálázhatók a thread poolinghez hasonlóan.
    - alkalmazások szolgáltatásokként futnak
        - automatikusan elindulnak, de el lehet indítani / le lehet állítani szolgáltatásokat manuálisan is
    - eseménykezelés
        - aszinkron
        - lehetővé teszi, hogy olyan komponensek várakozzanak az eseményekre, amelyek az esemény bekövetkezésének pillanatában még nincsenek is létrehozva
    - kontextus
        - objektum létrehozásakor létrejövő futásidejű tulajdonságok, amelyeken keresztül a rendszer nyilvántartja az objektum állapotát
        - api hívással (CoGetObjectContext) bizonyos tulajdonságok állíthatóak (pl. biztonság)
    - just in time aktiváció
        - object pooling: komponens megszüntetésekor nem törli azt a memóriából, hanem a későbbi létrehozáskor azt adja oda a kliensnek.
            - olyan helyen használatos, ahol a létrehozás-használat-megszüntetés folyamat ciklikusan fordul elő
        - JIT activation: régen használt szerver komponenseket ideiglenesen deaktiválja, majd egyből aktiválja egy másik kliens részére.
            - egy komponens több klienset is ki tud szolgálni (de nem egyszerre)
            - olyan helyen használatos, ahol a kliensek létrehozzák a komponenst és ahelyett, hogy megszüntetnék, ülnek rajta, és csak a program leállításakor szabadítják fel.
    - objektum konstruktor stringek
        - IObjectConstuctor interfész
        - description jellegű feladatokra
        - kommunikációs lehetőség a fejlesztő és az üzemeltető között
    - queued components
        - lehet, hogy a hívó már megszűnik, mire a hívottat létrehozák
    - biztonság
        - szerep alapú biztonság
        - szerepeket komponens iterfészekhez rendelünk, esetleg azok metódusaihoz is lehet. Így finom hangolható a biztonság.

- szinkronizáció
  - egy komponens csak akkor hívható meg, ha éppen nem csinál semmit, azaz a run-time a komponenst kritikus szakaszban futtatja
- tranzakciók
  - MTS integrálása

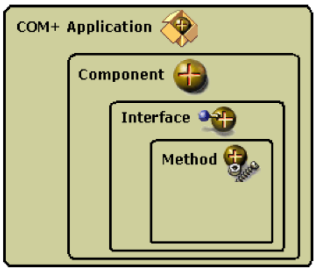
	COM	COM+
Tranzakció létrehozása	MTS-ben kell létrehozni a tranzakciót	Adminisztratív felületen a komponensre kell beállítani, hogy trben fog futni
DB módosítás	Adatbázisban	Adatbázisban
Más módosító függvények meghívása	MTS segítségével létre kell hozni a komponenst és meghívni	Simán, MTS nélkül
Hibakezelés, és tranzakciók nyugtázása, leállítása	Kezelni a hibákat, és ennek megfelelően MTS-ben commit vagy abort	Komponens dönt, hogy sikeres-e a tranzakció ráeső része, ha igen akkor egyesével commit vagy abort, majd a COM+ kiértékeli a végső commit vagy abortot.

  -

5.Ea

- COM+: COM alapú komponensek middleware szolgáltatásokkal kiegészítve
- .NET
  - nincs IDL, mert a fordító generálja
  - nem függ a registry-től-> nincs dll hell
- COM vs .NET

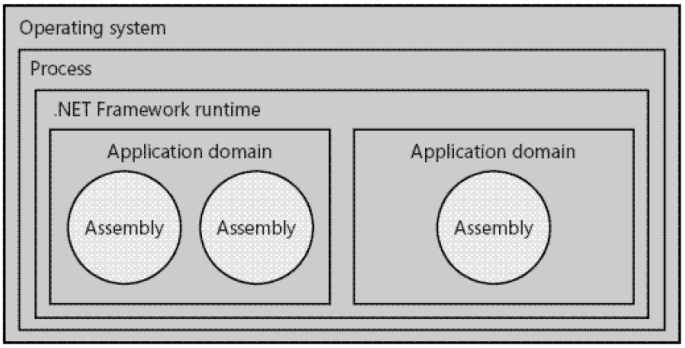
COM+ szintek (előző előadásból)



COM	**.NET**
Regisztráció	Önleíró komponensek
GUID	hierarchikus névterek
IDL fájlok	forráskódból mettaadatot generál a fordító
HRESULT	struktúrált exception-ök
IUnkown	Root objektum osztály

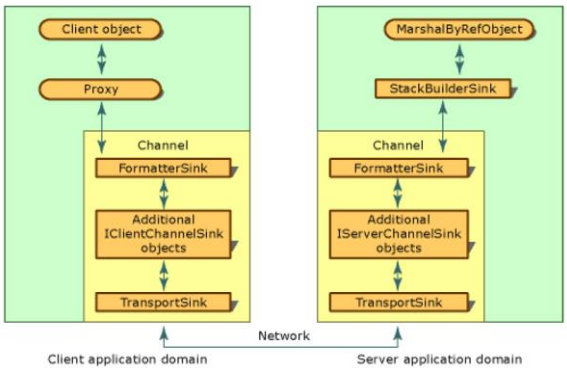
- COM és .NET együttműködés
  - .NET osztályok COM objektumokká alakítása
  - COM objektumok importálása
  - .NET osztályok COM+ szolgáltatásokat használnak
  - Könnyű rendszer hívások
    - Wrappereken keresztüli hívások
- .NET Enterprise Services

- osztályok a System.EnterpriseServices névtérben COM+ szolgáltatásokat csomagolnak be és könnyűvé teszik a COM komponensek építését
  - ServicedComponentből kell származtatni
- Assembly regisztráció
  - kézzel: RegSvcs.exe /fc MyAssembly.dll
  - dinamikusan: amikor a kliens példányosít egy serviced componentet.
  - programozottan: RegistrationHelper() segítségével
- .NET remoting
  - AppDomain: alkalmazástartományok közötti kommunikációra
    - alapból minden alkalmazásban létrejön egy default alkalmazástartomány, amiben fut a kódunk. De egy folyamatban lehet több is. Egymástól elszigetelten futnak, nem férnek hozzá egymáshoz.
    - lehet szabályozni egyes alkalmazástartományok jogosultságait
    - egy alk. tartományt el lehet különíteni egy alkalmazásból, ha nem kell. Egy betöltött assembly-t nem.



  - Szereplők
    - **Remotable Object:** távolról elérhető objektum. MarshalByRefObject osztályból kell származnia.
    - **Szerver:** Hostolja a remotable objectet.
    - **Kliens:** a szerveren hosztolt távoli objektumhoz kapcsolódva meg tudja hívni annak metódusait. A kliens a távoli objektumot egy helyi proxy-n keresztül látja.
      - a kliens és a szerver számára is rendelkezésre kell állnia a távolról elérhető objektum implementációjának
  - Kommunikáció
    - **Channel**: definiálja a kommunikációs protokollt. Ezen a csatornán zajlik a kommunikáció. Alkalmazás szinten kell regisztrálni a csatornákat. Legalább egy csatorna kell a szervernek.
      - IPC ~ inter process communication
        - csak egy gépen futó folyamatok között
        - egy gépes esetben ez a leghatékonyabb, nem kell socket kommunikáció feleslegesen
      - TCP: TCP / IP socket alapú megoldás
        - binary formattert használja
        - Gyorsabb, mint a HTTP, LAN esetben jó
      - http
        - SOAP formattert használ (XML alapú)

- **Activation mode**: a távoli objektum élettartamának kezelésének módja. Mikor jöjjön létre? Melyik kérést szolgálja ki?
    - Client Activation (CAO): a kliens kérésére jön létre az objektum a szerver oldalon.
    - Server Activation (SAO): a szerver irányítja az objektumok létrehozását.
        - SingleCall: minden kliens kéréshez új objektum jön létre, miután lefut a kérés az objektum megszűnik.
        - Singleton: a szerveren egy példány jön létre az objektumhoz, és ez szolgál ki minden típust.
- **Formatter**: azt határozza meg, hogy milyen formában sorosítódjon az adat a hálózati adatfolyamba.
    - Binary: bináris formátumba sorosítja az adatot. Gyors, hatékony.
    - SOAP: SOAP formátumba sorosít, XML alapú. Lassabb. http-n érdemes használni.



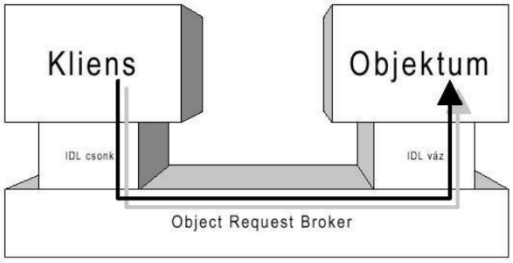
- Távoli objektumok címzése
    - szerver oldali csatorna által használt protokoll (http,tcp,ipc)
    - szerver címe
    - portszám
    - remoting alkalmazásnév
    - SAO esetben objectURI

- WCF ~ Windows Communication Foundation
    - újgenerációs technológia elosztott alkalmazások fejlesztéséhez.
    - kommunikációhoz kell
        - Address ~ Hol: végpont címe
        - Binding ~ Hogyan: kötés
            - Encoding + transzport + magasabb szintű szolgáltatások (biztonság, megbízhatóság stb.)
            - BasicHttpBinding, WSHttpBinding,NetTcpBinding
        - Contract ~ Mit
            - milyen interfészeket, metódusokat érhetünk el távolról
    - WCF szolgáltatások
        - szolgáltatás orientáció
        - iteroperabilitás
        - többféle üzenet minták

        - szolgáltatás metaadat
        - biztonság
        - tranzakciók
        - AJAX és REST támogatás
        - kiterjeszthetőség

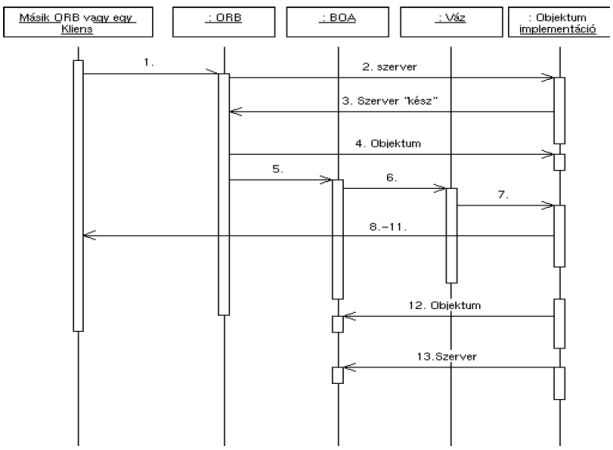
6.Ea - CORBA

- Component Object Request Broker Architecture
- IDL alapú interfész, heterogén környezet
    - minden metódus virtual public
    - nincsenek objektum változók
    - csak adatok vannak a struktúrában
    - speciális típusok
        - Any
        - DynAny
        - Oneway



- Automatizált funkciói
    - objektum helyének nyilvántartása
    - kapcsolat és memória management
    - paraméter kezelés
    - esemény és request demultiplexálás
    - hiba kezelés
    - szinkronizáció, konkurencia kezelés
    - biztonság
- Részei
    - Kliens
    - Facilities
        - Horizontális
            - UI, információkezelés, task management, system management
        - Vertikális
            - Egészségügy, pénzügy, telekommunikáció
    - Szolgáltatások: Middleware
        - életciklus, kapcsolat, naming, event, object query, konkurencia, licencing, idő, notification, tranzakció
    - ORB(Object Request Broker)
        - CORBA runtime
        - a kommunikációhoz nyújt szolgáltatásokat
        - API-t kínál
        - interfészt biztosít
        - kezeli a lokális és távoli hívásokat
    - Interface Repository
        - megkönnyíti az ORB-ok együttműködését
        - kérés paramétereinek típusellenőrzése
        - öröklési fák ellenőrzése
        - IDL fájlok kiváltása, mivel az IR minden információt tárol.
    - Objektum adapterek
        - implementációk regisztrálása
        - objektum referenciák létrehozása és generálása

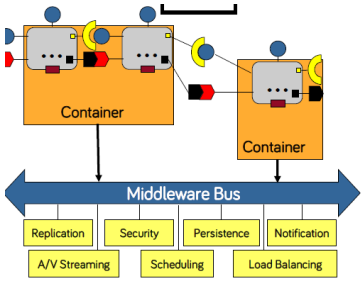
- implementációk aktiválása és deaktiválása
- biztonság garantálása a Security Service segítségével
- típusai
    - BOA ~ Basic Object Adapter
        - Négy féle kapcsolatot tesz lehetővé a szerver és az objektumok között
            - Osztott: olyan szerver, mely több objektumot tartalmaz
            - Állandó: ugyanolyan, mint az osztott, csak a szerver BOA-tól függetlenül aktiválandó
            - Egyedi: szerverenként legfeljebb egy objektum lehet aktív
            - Funkciónkénti szerver: A BOA minden funkció felhívás alkalmával elindít egy új szervert, amely a funkció végeztével leáll.
    - POA ~ Portable Object Adapter
        - Összetettebb és fejlettebb mint a BOA
        - általános objektum menedzsment célokra lett kifejlesztve
        - hierarchikusan több POA lehet egyszerre a rendszerben, egy jól definiált hierarchia segítségével
        - POA szereplők
            - Kliens: olyan kódrészlet, amely az objektumon valamilyen műveletet végrehajt.
            - Implementáció: IDL interfészben definiált objektumot megvalósító kód
            - Szerver: olyan kód, amely egy vagy több IDL interfész implementációját tartalmazza
            - Servant: Interfész implementációjának a példánya
            - ObjectID: POA-ban lokálisan értelmezett objektumazonosító. A POA ennek segítségével találja meg a megfelelő Servantot. Kliensek számára rejtett.



- ORB-ok közti kommunikáció
    - GIOP ~ General Inter Orb Protokoll

- egyszerűség, méretezhetőség, általánosság, architekturális függetlenség
- részei
    - CDR ~ Common Data Representation
        - általános adatábrázolási definíció
        - adatillesztés
        - byte sorrendiség
    - szállítási felételek
        - kapcsolat alapú protokoll
        - megbízható adattovábbítás
- GIOP üzenet típusok
    - Request Message (Kliens -> Szerver)
        - kérés továbbító üzenet
        - részei
            - GIOP fejléc
                - GIOP_version
                - byte_order
                - message_type
                - message_size
            - kérés fejléc
                - szolgáltatás kontextus található itt, benne kérés azonosító, obj azonosító, felhívandó funkció neve stb.
            - kérés törzs
                - a funkció összes in és inout irányú paramétereit tartalmazza
    - Reply Message (Szerver -> Kliens)
        - befejezett kérésre való reakciót továbbítja a kliens felé.
        - részei
            - GIOP fejléc
            - válasz fejléc
                - szolgáltatás kontextus, státusz kód
            - válasz törzs
                - amennyiben a státusz kód no_exception akkor az összes out és inout paraméterét tartalmazza
                - ha a státusz kód System_exception vagy User_exception akkor a kivétel szövegét tartalmazza
                - ha a státusz kód: Location_Forward, akkor automatikusan új címre továbbíthatja a kérést
    - Cancel Request (Kliens -> Szerver)

            - utasítja a szervert egy végrehajtás alatt álló RequestMessage vagy LocateRequest üzenet végrehajtásának törlésére.
                - részei
                    - GIOP fejréc
                    - kérésazonosító
                        - innen tudja a szerver, hogy melyik üzenet feldolgozását fejezze be
        - Locate Request (Kliens -> Szerver)
            - Ezen üzenet segítségével a kliens megtudja a szervertől, hogy az adott objektum referencia még érvényes-e, illetve, hogy a szerver képes-e feldolgozni az adott kérést.
        - Locate Reply (Szerver -> Kliens)
            - LocateRequest üzenetre válaszol a szerver ezzel az üzenettel.
        - Close Connection (Szerver -> Kliens)
            - Jelzi a szerver, hogy befejezi kapcsolatát az ORB-bal.
            - Ezt csak akkor teheti meg, ha nincs folyamatban levő kérés
        - MessageError (kétirányú)
            - bármelyik fél ezzel az üzenettel jelezheti bármilyen GIOP üzenet hibás voltát.
            - csak egy GIOP fejlécből áll
- CORBA hátrányok
    - Eltérő implementációk
    - Inkompatibilis megoldások implementációk között
    - Laza szabványosítás
    - Migrálhatóság nehézsége
- Skálázhatósági problémák
- Lehetséges megoldások a hátrányokra
    - CCM ~ CORBA Component Model
        - Komponensek becsomagolják az alkalmazás üzleti logikáját
        - Komponensek portokon keresztül kommunikálnak
            - Interfészek
            - Csatlakozási pontok
            - Event sinks
            - Attribútumok
        - Részei
            - Komponens szerver
            - CIF ~ Component Implementation Framework
            - Component packaging tools
            - Component deployment tools

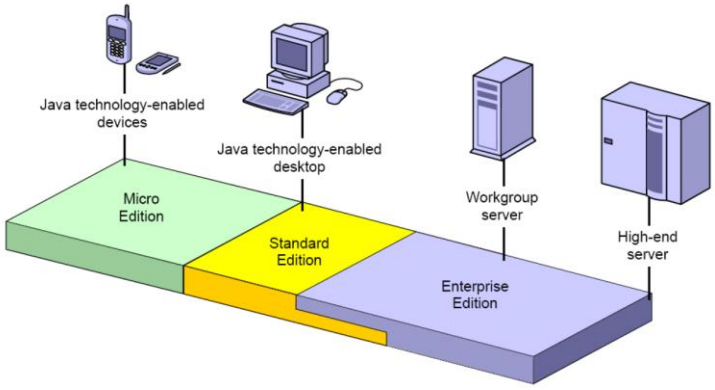


    - Container mechanizmus
    - Szabványos konfigurálási és telepítési mechanizmusok
    - Speciális interfészek támogatása
    - Kommunikációs pontok

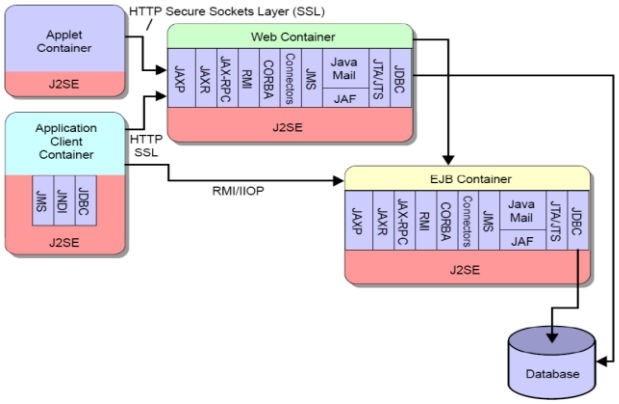
- CORBA komponens portok
  - Facets ~ providers: Interfészeket ajánlanak ki
  - Recepacles ~ uses: Szükséges interfészeket igényelnek, használnak
  - Event sources ~ előállt eventek
  - Event sinks ~ elfogyasztott eventek
  - Attributes ~ Konfigurálható tulajdonságok

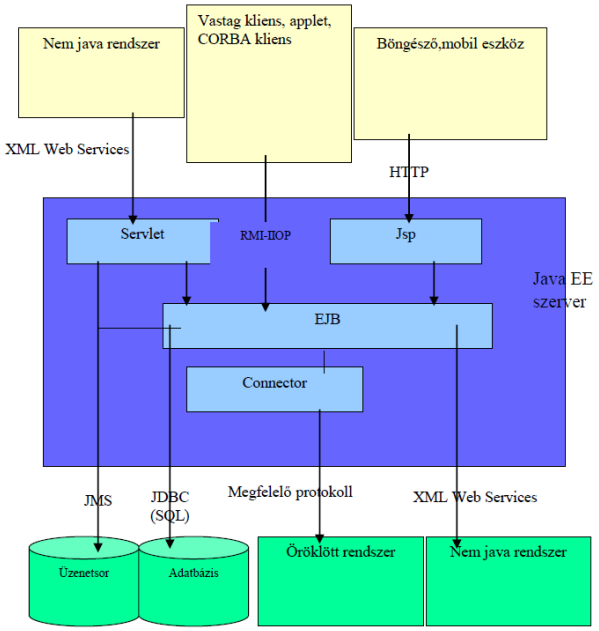
7.Ea JEE

- Java Enterprise Edition
- Vállalati elosztott rendszerek fejlesztéséhez
- Middleware szolgáltatásai
  - többszálúság
  - tranzakció kezelés
  - biztonság
  - perzisztencia
  - objektumok életciklusának kezelése
  - távoli metódushívás
  - skálázhatóság
  - terhelés kiegyenlítés
- 3 platform



- Containerek
  - ahhoz, hogy a JEE alkalmazások futni tudjanak container-be kell őket deployolni.
  - futtatókörnyezetet biztosít az alkalmazás számára
  - a containerek különböző szolgáltatásokat nyújtanak az alkalmazás számára
    - RMI,CORBA,MAIL





- APIk
  - JPA ~ Java Persistence Api: objektum-relációs leképezés
  - EJB ~ Enterprise JavaBeans: elosztott, üzleti logikai komponensek
  - JMS ~ Java Message Service: aszinkron üzenetkezelés
  - JTA ~ Java Transaction API: tranzakciókezelés
  - CDI ~ Contexts and Dependency Injection: függőséginjektálás
  - JAAS ~ Java Authenticaion and Authorization Service: biztonság
  - Webes technológiák
    - JAVA Servlet
    - JavaServer Pages
    - JavaServer Faces
  - Webszolgáltatások
    - JAX-WS~ Java API for XML-Based Web Services: SOAP alapú XML webszolgáltatásokhoz
    - JAX-RS~ Java API for RESTful Web Services: RESTes webszolgáltatásokhoz
- EJB
  - Szabványos felülettel rendelkező, elosztott, szerver oldali komponensek, melyek az üzleti logikát tartalmazzák.
  - EJB konténerben futnak, amely elfedi
    - hálózati kommunikációt
    - többszálúságot
    - tranzakció kezelést és más kényelmi szolgáltatásokat nyújt
  - Típusai
    - Session
      - Üzleti **folyamatokat** reprezentálnak (hitelkártya kezelés)
      - egyes használati eseteknek metódusokat feleltetünk meg
      - lehet **állapotmentes**, vagy **állapottal rendelkező**, vagy **singleton** (kizáró vagy)
        - Singleton
          - egy példány lesz egy EJB-ből, klaszterezett környezettben minden JVM-hez egy példány

        - Entity
            - Adatok reprezentálása a feladata
            - DB-vel tartják a kapcsolatot
            - Entitásokat reprezentálnak
            - ORM ~ Objektum relációs leképezést valósítanak meg
            - egy entity bean példány = az adatbázis egy sorának a memóriabeli cache-e.
        - Message-driven
            - Aszinkron üzenetkezelést végez
- Annotációk
    - Forráskódba illesztett metaadat
    - közvetlenül nem a program jelentését módosítja, hanem azt, hogy a különböző külső eszközök és libraryk hogyan kezeljék
    - JEE-ben a telepítésleírók szerepét veszik át
        - alkalmazásszerver-gyártó feladata az annotációkat értelmezni telepítéskor
        - de a telepítésleírók továbbra is használhatók, és felüldefiniálják az annotációkat ha vannak.
    - Szintaxis: *@AnnotációNeve(param1=érték1, param2 = érték2…)*
    - beépített annotációk
        - @Override: gépelési hibák elkerülése
        - @Depracated:elavult metódusok megjelölésére
        - @SurpressWarning: warningok elrejtését végzi
    - EJB annotációk
        - állapotkezelés definiálása
        - singleton-ok definiálása
        - függőség injektálás
        - aszinkron metódusok
        - interceptorok
- JEE szerepek
    - **komponens fejlesztő**: alkalmazásszerver független komponenseket ír
    - **application assembler**: komponensekből összeállít egy alkalmazást, ami még mindig gyártófüggetlen
    - **deployer**: telepíti az alkalmazást a konkrét alkalmazás szerverre
    - **rendszeradminisztrátor**: monitorozza a futó alkalmazást, és ez alapján optimális teljesítményre hangolja
    - **eszközgyártó**: előző tevékenységek elvégzését támogató eszközöket gyárt
    - **alkalmazásszerver-gyártó**
- **JEE vs COM+/.NET**

Funkcionális hasonlóság

Windows	.NET	J2EE
Win 32	Intermediate Language (IL)	Java byte code
VB/MFC runtime	Common Language Runtime (CLR)	Java VM
COM component	.NET component	Java Bean
MTS/COM+	COM+/.NET Enterprise Services	EJB
ASP	ASP.NET	JSP
ISAPI Extensions	HTTP Modules	Servlets
ADO	ADO.NET	JDBC/JDO
DCOM	.NET Remoting	RMI/IIOP
SOAP	SOAP	(SOAP)
VB/MFC	Windows Forms	Swing
ActiveX	Windows Forms component	Applet
C++, VB, Java, Delphi	C#, C++, VB.NET, Java, COBOL, Eiffel, ...	Java

- **.NET**

    - Com+ szolgáltatások
    - Context, biztonság,tranzakciók, pooling, lazán-csatolt események, aszinkron események
    - Operációs rendszerrel integrált. Optimalizált a sebessége
- **JEE**
    - EJB Szerver / Konténer menedzseli az EJB-ket
    - Context, biztonság, tranzakciók, pooling, menedzselt perzisztencia, aszinkron események
    - JEE app szerver része – ez egy plusz réteg az OS felett
- **TEHÁT NAGYON HASONLÓ A KETTŐ**
    - átjárás a kettő között
        - IIOP.NET: remoting kiegészítése új csatornával (Java is hozzáfér IIOPn keresztül .NET szerverhez)
        - J-integra for .NET: remoting protokoll Java implementációja
        - **webszolgáltatásokkal is átjárható a kettő**

9.Ea – Szöveges szakterületi nyelvek

DSL ~ Domain specific language: szakterületi nyelv

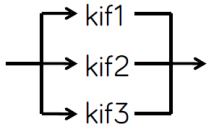
- Programozási, vagy specifikációs nyelv korlátozott kifejezőerővel, egy konkrét szakterület problémáinak és megoldásainak leírásához.
- megjelenési formája lehet szöveges, illetve grafikus
- Tulajdonságai
    - szakterület absztrakciós szintjén dolgozhatunk, és szakterületi fogalmakkal fejezhetjük ki a problémát
    - szakterületi nyelven írt programok tömörek, jól dokumentálják saját magukat és újrafelhasználhatóak
    - növelik a produktivitást, megbízhatóságot, karbantarthatóságot és a hordozhatóságot, segítik a tesztelést
- Hátrányai
    - szakterületi nyelv tervezése, megvalósítása, karbantartása költséges
    - felhasználóit be kell tanítani a nyelv használatára, ami költségekkel jár
    - határainak megállapítása nehéz
        - ha túl általános, akkor nem növeli az absztrakciót
        - ha nem elég általános, akkor nem tudunk vele mindent kifejezni
- Szöveges szakterületi nyelvek
    - PictureProcessor nyelv
        - képek feldolgozásánál
        - input: képek neve és opciók a feldolgozáshoz
            - kimenet felbontása, neve, formátuma, minősége
        - egyszerre sok fájl
            - ne kelljen egyesével felsorolni a fájlneveket, legyen egy from és to paraméter, és a feldolgozó találja ki a közbülső fájlneveket, ha azokban csak egy szám különbség van.
            - opciók összevonása: $$$ jelentse az eredeti fájl nevét a kimeneti fájlnévnél
        - később több igény is felmerülhet
        - *Process –from=DSC3456.jpg –to=DSC3465.jpg –size=1024 –out=JPG -outfile=$$$s.jpg*
    - SQL

        - egy nyelv adatbázisok kezeléséhez
        - adatok struktúrája, adatok közti kapcsolatok
    - XAML:Grafikus megjelenítés specifikálása WPF-hez
    - HTML: szöveg formázás, tagolás, adatbeviteli űrlapok
    - CSS: HTML kiegészítése, megjelenítés testreszabása
    - LaTex: dokumentumszerkesztés
    - LOGO: rajzoló programnyelv
    - Verilog: áramkörök tervezéséhez
    - CSound: hangok, audioeffektek kezelésére
- nyelv jellege szerinti csoportosítás
    - Internal DSL
        - általános célú programozási nyelv, spec módon használva
        - nyelvelemek közül csak néhányat használunk
        - pl: script nyelvek
    - External DSL
        - saját nyelv, nem az alkalmazás programozási nyelve
        - saját szintaxis, vagy egy más nyelv szintaxisa
        - pl: PicProcessor nyelv, SQL
- programozási mód szerint
    - deklaratív: végrehajtás módja helyett a kívánt eredmény a fontos: SQL
    - imperatív: végrehajtás módját adjuk meg, LOGO
- DSL megadási módszerek: XML-lel
- Szöveges metanyelvek ~ EBNF (Enchanced Backus-Naur Form)
    - metanyelv olyan nyelv, ami más nyelvek definiálására alkalmas
    - állításokat és szabályokat fogalmaznak meg a definiálandó nyelvre vonatkozóan
    - építőelemei
        - **terminális szimbólumok:** cél nyelv részei pl: **int**
        - **nem terminális szimbólumok:** szintaktikai kategóriát írnak le, amely több nyelvi szimbólumra utal: **number**
        - **metanyelv operátorok:** operátor ami a terminális és nem-terminális szimbólumokon van értelmezve pl: or
    - produkciós szabályokból (egyenletekből) áll
        - boolean = „True” | „False”
        - boolean = nem terminális, true és false terminális szimbólum.
        - metanyelv operátorok a „=” és „|”

Konkatenáció: kif1 → kif2 → kif3 kif1, kif2, kif3

property = name,":", type;

Választás:

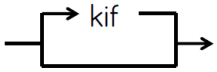


(kif1| kif2 | kif3)

visibility = "+" | "-" | "#" | "~";
multiplicity = "1" | (lower-bound, ".. ", upper-bound);

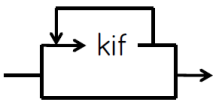
Kivonás: not kif1 - kif1

number = Digit - „0”;

Opcionális elem: 

[kif]

property = [visibility], name, [":", type],

Ismétlés: 

{ kif }

number = ["-"] , digit , { digit } ;
identifier = character , {character | digit } ;

- Csoportosítás:

<expr> ::= <expr> + <term> | <expr> - <term>

<expr> ::= <expr> (+|-) <expr>

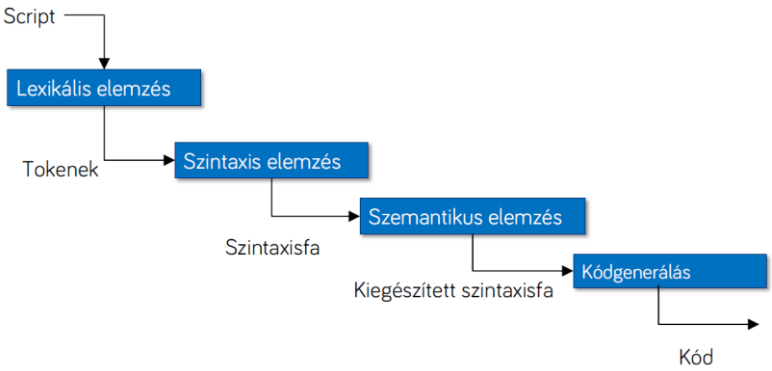
- Opcionális elem
- Ismétlés szabályozása:

<binary-seq> ::= <binary-digit> |
<binary-digit> <binary-sequence>

<binary-seq> ::= { <binary-digit> }+

```
Program = {Command};
Command = (Pen | Move | Repeat), („„|" \n" );
Pen = „PENUP"  | „PENDOWN" ;
Move = („FWD" |" LEFT" |" RIGHT" ), „ „, Num;
Repeat = „REPEAT" , „ „ , Num, „[" ,{Command}, „]" ;
Num = Digit - „0" , {Digit};
Digit = 0|1|2|3|4|5|6|7|8|9;
```

- Szakterületi nyelvek feldolgozása
  - Interpreter
    - memóriában
    - utasításonként
    - gyorsan indul, lassan fut
    - első hiba után leáll
    - debug könnyű
  - Compiler
    - kódot generál
    - egész programot egyben dolgozza fel
    - lassan indul, gyorsan fut
    - végén jelzi a hibákat
    - debug nehéz
- Fordítók működése

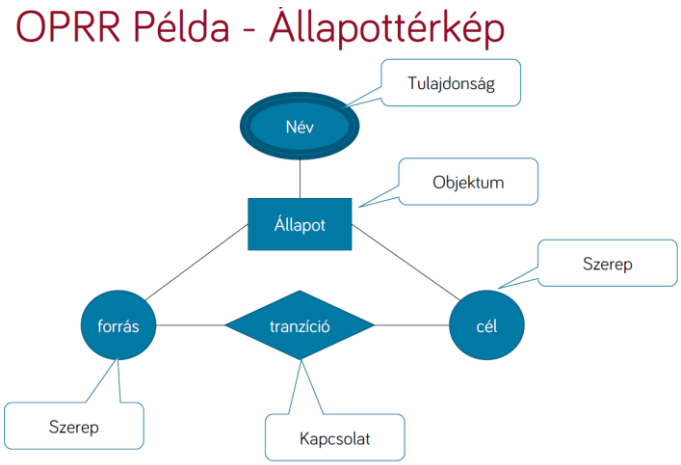


  -
  - Lexikális
    - Szövegből tokeneket készít
    - token pl: operátor, azonosító
  - szintaxis
    - nyelvtani szabályok alapján szintaxis fát épít
    - tokeneket fába rendezi
  - szemantikus
    - elemzi, ellenőrzi, kiegészíti, átformálja a szintaxis fát
    - típusellenőrzéssel
  - kódgenerálás
    - szintaxisfa bejárásával
    - nem binárist generál hanem: C,C#,Java kódot
- Mercury ST: script nyelv

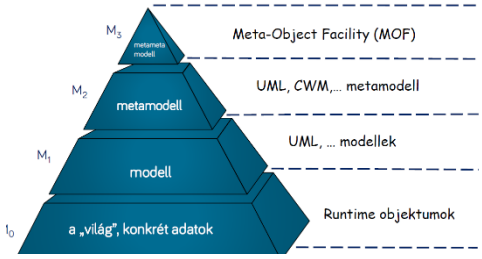
10.Ea – Vizuális nyelvek

- Szoftverfejlesztés legproblémásabb lépése a szoftverfejlesztő-szakértő kommunikáció.
- ha szakterületi jeleket használjuk, a kommunikáció sokkal könnyebb
- grafikus modellekkel a kommunikáció sokkal könnyebb.
  - kisebb a tanulási idő
  - könnyebb és gyorsabb az információ átadás
- ***részei***
  - Nyelv struktúra és kiegészítő kényerek -> Absztrakt szintaxis
  - Megjelenítés -> Konkrét szintaxis
    - ahogy megjelenik a felhasználó számára
  - Struktúra jelentése -> Szemantika
    - nyelv / modell jelentése valamilyen formalizmust követve
- UML
  - vizuális modellező nyelv
  - programozási nyelvtől független
  - kiterjeszthető, szabványos
  - saját metamodelljével leírható
  - UML Profile
    - UML által kínált nyelvek specializálására
    - általános elemek specializációját írja le, sztereotípiákkal, tag-ekkel és kényszerekkel
    - nem mond ellent az eredeti specifikációnak
    - pl: SysML ~ Systems Modeling Laguage
      - általános modellező nyelv rendszertervezéshez
- Metamodellezés

    - példányosítás reláción alapuló absztrakt folyamat, ami egy szakterület absztrakt szintaxisát formálisan adja meg
    - definiálja
        - modell elemek
        - kapcsolatok közöttük
        - attribútumok
        - kényszerek és szabályszerűségek
    - független a konkrét szintaxistól
    - Metamodellező nyelv
        - modellek absztrakt szintaxisát a metamodellek definiálják
        - maga is egy spec szakterületi nyelv
- Entity-Relationship Diagram
    - leginkább DB modellek tervezésére
    - ORRR: ER kiterjesztése
        - Objektum: önmagában létező valami
        - Property: jellemző adat
        - Kapcsolat: két vagy több objektum közt
        - Szerep: az objektum és kapcsolatának nevesítése



- MOF ~ Meta Object Facility
    - szemantikailag jól definiált modell a metamodellek leírására
        - metainformációk kezelésére szolgáló keretrendszer
        - formális alapot ad a metamodellezéshez
    - szintjei
        - metametamodell ~ MOF
        - metamodell
        - modell
        - runtime objektumok, konkrét adatok



Szint	Leírás	Példák
Metameta-modell	Metamodellezési architektúra Metamodellező nyelvek létrehozására.	MetaClass, MetaAttribute, MetaOperation
Metamodell	Metametamodell példánya. Szakterületi és modellezési nyelvek létrehozására.	Class, Attribute, Operation, Component
Modell	A metamodell példánya, konkrét szakterületi modellek.	Product, Unit Price, Customer, Sale, Detail
Objektumok, adatok	A modell példánya, konkrét, adatokkal kitöltött modell	<Chair>, <Desk>, $100, $200

    - Építőelemek
        - Osztály: fogalmak reprezentációjához
        - Kapcsolat: két osztály közötti viszony
        - Csomag: csoportosítás megadásához
    - Variánsok
        - EMOF (Essential MOF): alap funkciók, egyszerű metamodellekhez
        - CMOF (Complete MOF) teljes verzió
    - Metaadatok sorosítása
        - XML Metadate Interchange (XMI)
    - Mély példányosítás: ha n. szinten modellezzett információ elérhető az n+x (x>1) szinten is
- VMTS ~ Visual Modeling and Transformation System
    - ne korlátozzuk a modellezési szintek számát
    - n szintű modellezési hierarchia
    - ős metamodell ~ atom
    - háromféle kapcsolat: tartalmazás, öröklés, asszociáció

11.Ea - Kényszerek

- Kényszer: egy megszorítás a metamodell egy vagy több elemén, értékén.
- OCL – Object Constrait Language
    - egy kényszerdefiníciós nyelv
    - tulajdonságai
        - kényszerek deklaratívak: azt adják meg, hogy mi a helyes, és nem azt, hogy mit kell tenni
        - kényszereknek nincs mellékhatásuk: kiértékelésük nem változtatja meg a rendszer állapotát
        - kényszerek formális szintaxissal és szemantikával rendelkeznek: értelmezésük egyértelmű, és automatizálható
    - kontextus: az a modell elem, amire a kényszert definiálták
        - osztály, interfész, adattípus, művelet, példány
    - kontextus típusa: annak a modellelemnek a típusa, amire a kényszert kiértékelik
    - kontextus példány: a konkrét modellelem, amire a kényszert kiértékeljük
        - self kulcsszóval hivatkozunk rá
    - kifejezéstípusok
        - invariáns: A felnőtt ember kora nagyobb, mint 18.
            - metamodell elemre definiált
            - logikai kifejezés, amely a meta modell elem minden példányára minden időpillanatban igaznak kell lennie.

```
context Customer
inv: self.name = 'Edward'

context Customer
inv: age >= 18

context CustomerCard
inv checkDates:
validFrom.isBefore(goodThru)
```

- elő és utófeltétel: A túra elején 3 kiló étel volt nálunk, a végén 0.
  - műveletre definiált
  - előfeltétel: a művelet elvégzése előtti utolsó pillanatban igaz feltétel
  - utófeltétel: a művelet elvégzése utáni első időpillanatban igaz feltétel
- kezdeti értékek: Születéskor minden gyerek szeme kék.
  - attribútumra vagy asszociációra definiálva
  - egy érték, amelyet felvesz a kezdeti állapotban az attribútum vagy az asszociáció.

```
context CustomerCard::transactions :
              Set(Transaction)
init: Set{}

context CustomerCard::valid : Boolean
init: true
```

- származtatott értékek: A végső érdemjegy a ZH és a vizsga átlaga.
  - attribútumra vagy asszociációra definiálva
  - nem önmagában létező érték, mindig más elemek segítségével definiálják

```
context CustomerCard::printedName
derive:
owner.title.concat(owner.name)

context CustomerCard::goodThru
derive: if ( owner.isMale='male' )
        then validFrom.addYears(5)
        else validFrom.addYears(3)
        endif
```

12.Ea

- Szöveges DSL
  - Konkrét szintaxis: A nyelv, ahogy megjelenik a programozó felhasználó számára
    - a szöveges DSL jellemzően magába foglalja a nyelv konkrét szintaxisát

- Vizuális DSL
    - metamodell kiegészítések
    - kódolás
- Szemantika: ☹……