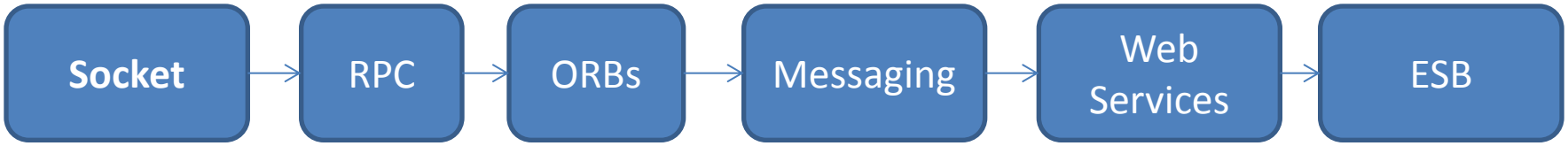
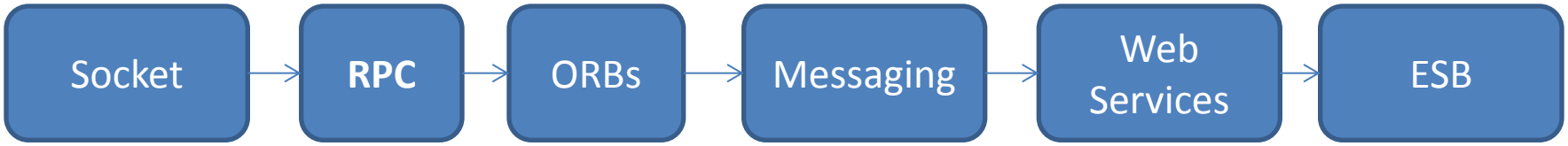
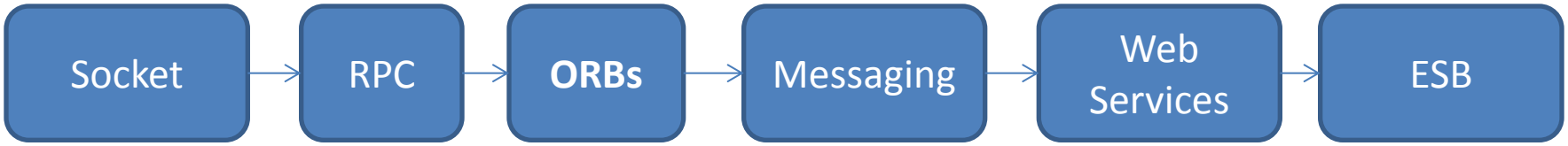
Információs rendszerek integrációja



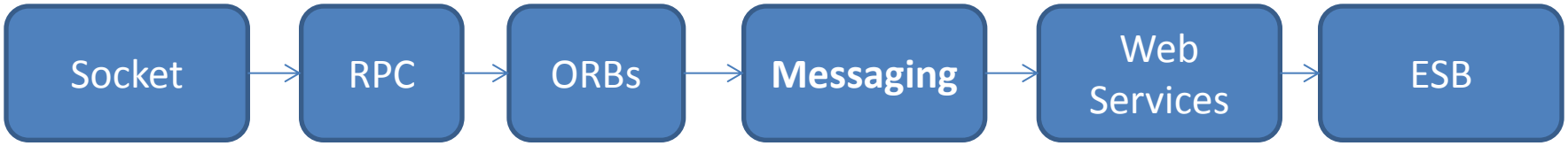
- TCP socket alapú integráció
  - Valós idejű adattovábbítás
    - Általában bináris adattovábbítás, de szöveges XML is lehet
    - Nincs közvetlen eljárásmegosztás („kézzel” kell megvalósítani)
    - TCP vagy UDP kapcsolat is lehetséges (UDP esetén többszörös a sebességnövekedés (streamelés))
  - Állandó vagy on-demand csatorna
    - A kapcsolódási időveszteség miatt érdemes állandó kapcsolatot fenntartani, periodikus életjel küldéssel kombinálva. Hiba esetén automatikus újracsatlakozás.
    - UDP esetén lehet plusz TCP csatorna a szinkronizáció miatt.
  - A leggyorsabb átvitelt eredményezi
    - Ahol a tranzakciók száma eléri a másodpercenkénti ~20 tranzakciót ott már érdemes alkalmazni. (50ms / sec átvitel)



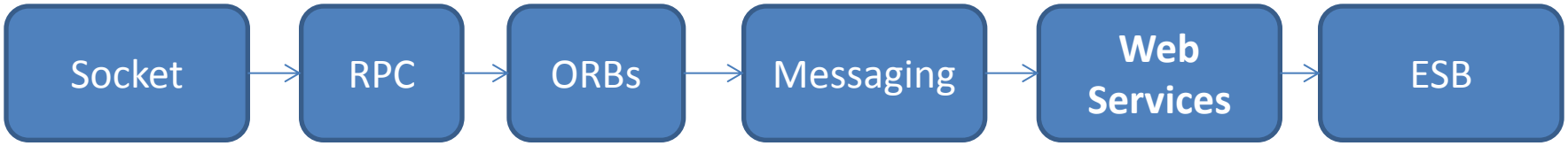
- Remote Process Call – távoli eljáráshívás
  - Klasszikus kliens-szerver programozás. Az alatta elhelyezkedő socket megoldás rejtve marad a fejlesztőtől.
  - A csatorna már nemcsak egyszerű adattovábbításra használható, hanem eljárások távoli végrehajtását is beépítve (API szinten) támogatja.
  - Eljárások interfészleírása
    - Szükség van a távoli eljárások valamilyen formában való leírására. XDR (External Data Representation szabvány) segítségével
  - Platform függetlenség
    - Megoldható a platform független használat az interfészleírók segítségével



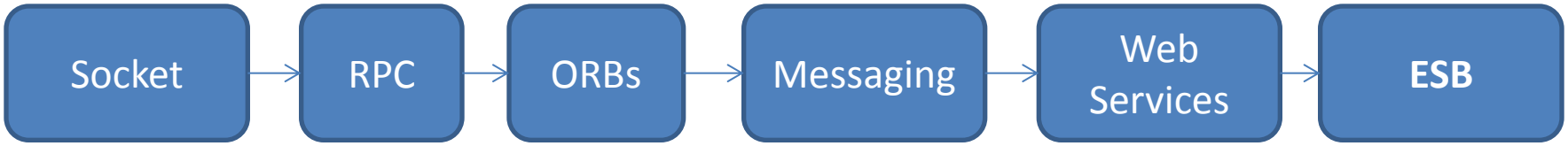
- Object Request Broker
  - Először vezeti be az objektum orientált programozás paradigmát az integrációban, és kiterjeszti az objektumok fogalmát a távoli – más alkalmazások által is használható - objektumokra.
  - Távoli objektumok kommunikálhatnak egymással
    - Megoszthatnak eljárásokat és adatokat egyaránt
  - Ismert megvalósítások: a Corba és a Java-RMI
    - Marshalling/Unmarshalling fogalmak megjelennek, ami az átadott paraméterek/adatok platform független szerializációját jelentik. Corba eljárással különböző platformok is könnyen kommunikálnak egymással
  - Megjelenik a nyelv független service-interface
  - Megjelenik a Service Registry kezdeti koncepciója
    - A szolgáltatások egységes módon kereshetőek, valamilyen „adatbázisból”.
  - WebSphere és Jboss alkalmazás szerverek ORB technológián alapulnak



- Üzenetkezelő rendszerek
  - Az ORB-val párhuzamosan asszinkron rendszerek alakultak ki.
    - A háttérben itt is socket alapú kommunikáció van, de a megoldás segíti a skálázhatóságot.
    - A skálázhatóság annak az eredménye, hogy az üzenet elküldésekor nem kell megvárni a választ (ezért asszinkron), hanem a feldolgozás folyamatos.
    - Megjelentek az üzenetsorok, amelyek a küldésért és fogadásért felelősek.
    - Ez a közvetett módszer elősegítette a rendszerek laza kapcsolódását.
    - Megvalósítható a garantált üzenetkézbesítés, ugyancsak a köztes tároló (üzenetsor) miatt.
    - Szinkron üzenetküldés szimulálható



- Webszolgáltatások
  - XML-en alapuló kommunikáció, ami egyesíti az RPC, ORB, üzenetküldés előnyeit
  - Az üzenetek és válaszok alacsony szinten is XML szövegként lesznek szállítva
    - Előnye, hogy a kommunikáció könnyen követhető, mert nem bináris adatot továbbítanak
    - Hátránya: az XML feldolgozás erőforrás-igényes
  - WSDL (Web Service Definition Language) alkalmazása az interfészleírásra
    - Nyelv, platform, alkalmazás kiszolgáló független
  - UDDI (Universal Description, Discovery and Intergration)
    - Registry alkalmazása a szolgáltatások felkutatásához, leírásához és integrációhoz
  - Az üzenetek továbbításának szabványos formátuma a SOAP
    - XML-RPC egy egyszerűbb megoldás a SOAP helyett



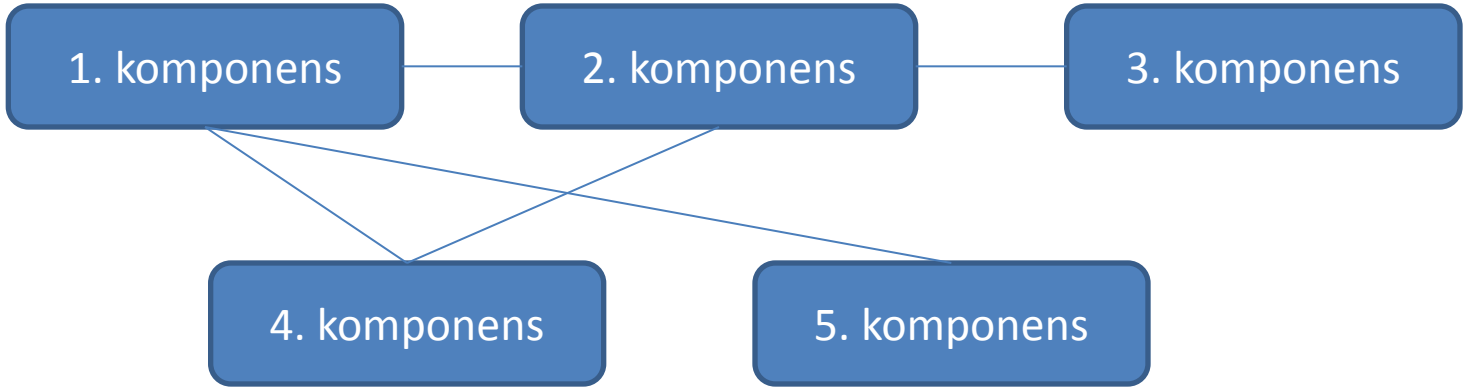
- Enterprise Service Bus
  - A Webszolgáltatások nem oldják meg hatékonyan a heterogén rendszerek összekapcsolását.
    - Nem oldják fel a kommunikációs protokoll eltéréseket, azaz ha az integráció során a szolgáltatás által kínált protokoll különbözik a felhasználó alkalmazás kommunikációs módszerétől.
  - Protokoll és üzenet transzformáció
    - message routing – kontextus és tartalom alapján
    - QoS (Quality of Service) – teljesítmény, megbízhatóság, biztonság
    - Data enrichment – automatikus további adatokkal való kiegészítés

SOA – Service Oriented Architecture

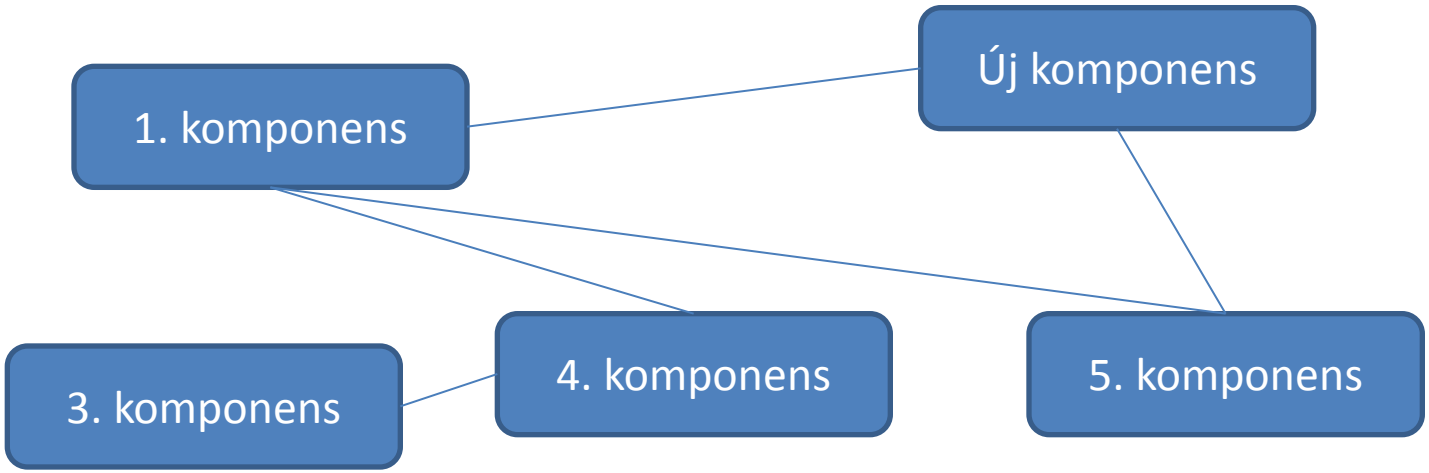
- SOA a gyorsan változó piaci környezetre ad IT választ
  - Korábban a szoftvereket nem kellett olyan gyakran módosítani. Az IT követelmények fix-ek voltak.
  - A termékek csökkenő életciklusa gyorsabb változást eredményez az IT-ben is.
  - Integrációs problémákat idézhetnek elő:
    - Összeolvadás: két vállalat egybeolvad
    - Akvizíció (felvásárlás): egyik vállalat felvásárolja a másikat
    - Piaci feltételek gyors változása
    - Új technológiák megjelenése – marketing és sales aktivitások folyamatos változása. (pl. facebook marketing)
    - Üzleti kapcsolatok változása: egyre több és gyakrabban változó üzleti partner jelenik meg.

SOA – Service Oriented Architecture

- Megoldás-e a SOA?
  - SOA agilis IT rendszereket hangsúlyoz, a komponensek újrahasznosításán keresztül.
  - Ebben a rendszerben a komponensek nem speciális üzleti problémákat, hanem általános funkcionalitást oldanak meg.
  - Változó üzleti környezetben nem kell új fejlesztés, hanem, a komponensek újrakonfigurálhatóak, összekapcsolhatóak, integrálhatóak az aktuális üzleti szükségletek alapján.



A fenti rajzon szoftverkomponensek, egy olyan lazán kapcsolt (loosely coupled) konfigurációja látható, amely megfelel egy adott üzleti követelménynek, valamilyen időintervallumban. Új komponens megjelenésével, a rendszert nem kell újra megvalósítani, elég a konfigurációt és az elrendezést megváltoztatni. Ezzel a módszerrel



- Újrahasznosított komponensek előnyei
  - Fejlesztési és tesztelési időmegtakarítás. Egy komponens jól definiálható input és outputtal rendelkezik, megfelelő tesztesetek átgondolása egyszerűbb. Korábban már tesztelt komponens újrahasznosítása kevesebb kockázatot jelent.
  - A redundáns kód kiküszöbölésével, a sokkal következetesebb funkciókkal és adatokkal rendelkezünk.
  - Könnyű karbantarthatóság, hatásvizsgálat (impact analysis) , mivel a változások komponensekre korlátozódnak.

Enterprise Software

- Nagyobb vállaltok számára tervezett szoftver, amelyek rendelkeznek saját belső szervezettel, folyamatokkal és üzleti modellel. Figyelembe veszi mind a részlegek közötti függőségeket és a külső üzleti kapcsolatokat, pl a üzleti partnerek és külső beszállítók
- Ezért egy információs rendszerben nagyszámú követelménynek kell teljesülnie. Sajnos a gyakorlatban sok követelmény részleges, nem tisztázott vagy más módon ellentétben vannak egymással
- Továbbá a követelmények folyamatosan változnak a piaci feltételek és szervezeti változások miatt
- A fenti tulajdonságok miatt egy információs rendszer nagyon komplex rendszer
- A szoftverfejlesztők számára viszont előnyös, hogy az alkalmazott üzleti logika általában nem tekinthető túl bonyolultnak, összehasonlítva egy beágyazott rendszer kódjával. Hasonlóan az adatszerkezetek sem annyira bonyolultak mint egy térinformatikai rendszer adatszerkezetei.

Enterprise Software

- Az üzleti adatok és más tartalmak életciklusa viszonylag nagy.
- Sokrétű technológiákat alkalmaznak, pl. különböző alkalmazás kiszolgálók és sokféle alkalmazások.
- A funkcionális követelmények állandóan változnak
- A rendszert felhasználók száma nagyon nagy is lehet. (több százezer)
- Az érintettek száma is nagyon nagy lehet, pl. különböző IT projektek, IT üzemeltetés, különböző üzleti egységek.

Ha összehasonlítjuk egy desktop alkalmazással (pl. szövegszerkesztő) azt lehet mondani, hogy a fejlesztő csoportok sokkal inhomogénebbek, az adatok sokszor az alkalmazáson túlmutatnak, átmeneti jellegűek.

Integráció

- Ha az információs rendszer különálló alkalmazásokból áll, nehéz átfogó és egységes képet adni a funkciókról és az adatokról. Minél nagyobb egy szervezet, annál több alkalmazást használnak, a kép egyre bonyolultabb. A probléma megoldására az alkalmazásoknak kommunikálniuk kell egymással, meg kell osztaniuk adataikat és eljárásaikat.
- Ezzel elérhető az adatok redundanciájának mérséklése.
- **Azt a szoftverfejlesztési folyamatot, amelyben olyan új alkalmazások fejlesztésével vagy meglévő alkalmazások átalakításával foglalkoznak, amelyek képesek megosztani adataikat és eljárásaikat, szoftver integrációnak nevezzük**.

Software Architecture

- Egy kisvállalkozás vagy szervezet informatikai igényeit általában kevés alkalmazás ki tudja elégíteni. Ezeket az alkalmazásokat könnyű menedzselni, de ahogy a szervezet mérete növekszik és az alkalmazások száma folyamatosan nő, átfogó tervezés szükséges, hogy a káoszt elkerüljük. **Ezt az átfogó tervezési stratégiát szoftver architektúrának nevezzük.**
- A szoftver architektúrát úgy kell tervezni, hogy mind a jelenlegi, mind a későbbi követelményeknek meg kell felelnie.
- Korábban akkor fejlesztettek egy üzleti alkalmazást amikor igény merült fel. Természetes, hogy elkülönülve működtek, nem is volt igény az integrációra.

Loose coupling (laza összekapcsolás)

- A laza összekapcsolás kulcsfontosságú fogalom a SOA típusú integrációs mintákban. Azért célszerű alkalmazni, szoftverfejlesztési szempontból ezzel minimalizálható a kód-változtatások száma. Üzleti szempontból a laza kapcsolatok alkalmazásával a leggyorsabb a szükségletekhez való igazodás.
- Egy elosztott rendszerben kapcsolódás sokféle szinten történhet.
  - Socketen keresztüli közvetlen hálózati kapcsolat szoros összekapcsolást (tight coupling) eredményez. A közvetítő, maga a hálózat.
  - MOM (message oriented middleware) rendszerekben a közvetítő a hálózat felett egy üzenetsor, ami már laza kapcsolatot eredményez.
  - Egy RPC stílusú kapcsolat ugyancsak szoros kapcsolatot eredményez, mivel a közvetítő ugyancsak végeredményben egy socket.
- Általában az aszinkron hívásokat laza kapcsolatnak értelmezhetjük
  - De csak akkor ha MOM rendszer a közvetítő! Aszinkron kapcsolatot szimulálni lehet RPC hívásokkal is, egyszerűen úgy, hogy rögtön visszaadjuk a vezérlést a hívó félnek, és egy szálat indítunk a hívott oldalon a feldolgozáshoz. Ezzel a kapcsolat még nem lesz „laza”, mivel ilyen esetben is pl. mind a két alkalmazásnak futnia kell a korrekt működéshez.
- Az összekapcsolódás következő szintjén az interfészek szemantikáját is meg kell vizsgálni.
  - Ha az interfészek explicit típusokat és metódusokat alkalmaznak akkor még mindig szoros kapcsolódásról beszélünk, mert ha megváltozik az interfész valamelyik oldalon, a helyes működés már nem biztosítható. A módosítás továbbgyűrűzhet más alkalmazásokra is.
  - Egy objektum orientált elosztott rendszerhez OO stílusú bejárást kell fejleszteni komplex objektum fákhoz. Így érdekességképpen egy RPC stílusú összekapcsolás lazább kapcsolatot jelent, mint az OO.

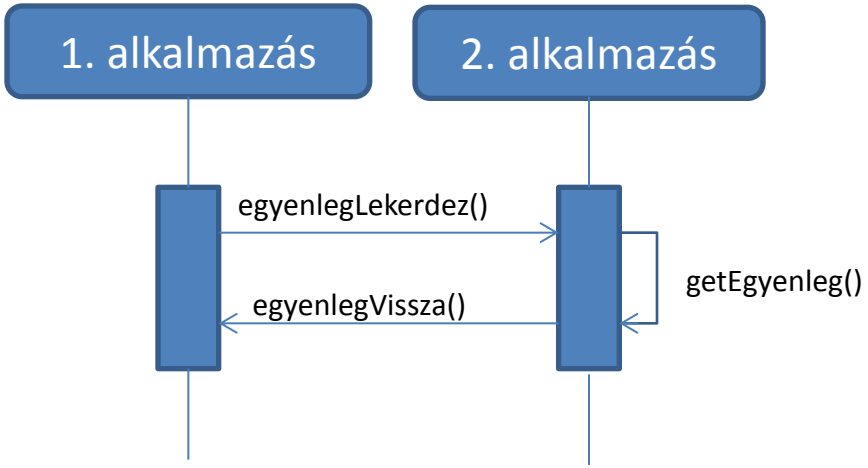
Loose coupling (laza összekapcsolás)

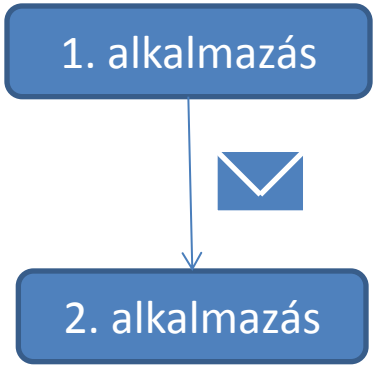
- Az összekapcsolás foka (degree of coupling) szempontjából meg kell vizsgálni, hogy a rendszer hogyan vezérli a folyamat logikát.
- Központi folyamatvezérlés szoros kapcsolatnak tekinthető a tranzakciók és a különböző alfolyamatok szempontjából.
  - ERP és CRM rendszerek ilyen szempontból monolitikusnak tekinthetőek
  - A kapcsolat már lazábbnak tekinthető ha elosztott B2B rendszereket is alkalmazunk, mert a központi folyamatvezérlés már elosztottá válik
- Az utolsó tulajdonság, amit érdemes figyelembe venni az, hogy a kliensek hogyan állapítják meg, hogy a kiszolgáló hol helyezkedik el?
  - Ameddig a statikusan kapcsolt alkalmazások szoros kapcsolatnak tekinthetőek, addig a dinamikus szolgáltatás felderítés (registry - és, névszolgáltatás alkalmazás a) már „lazábbra” veszi a kapcsolatot.

Faktor	Laza kapcsolat	Szoros kapcsolat
Fizikai összekapcsolás	Indirekt kapcsolat közvetítő segítségével	Közvetlen összekötés
Kommunikációs stílus	Aszinkron	Szinkron
Rendszertípus	Gyengén típusos rendszer	Erősen típusos rendszer
Interakciós minták	Elosztott logika	Központosított logika
Szolgáltatás felderítés	Dinamikus	Statikus (beégetett)

Interface és Payload szemantika

- A kliensek alapvetően kétféle módon határozzák meg, hogy milyen tevékenységet végeznek a kiszolgálón
  - Interfész definiálja, hogy milyen tevékenységek lehetségesek (hagyományos módszer)
  - Maga az üzenet tartalmazza a tevékenység (pl. fejlécben)





Adatmegosztás (Data Sharing)

- Az adatmegosztás a legrégebbi célja az osztott rendszereknek
- Először az adatmegosztást valósították meg, azután jött a funkciók megosztása
- Korábbi tanulmányok során három technikát is megismerhettünk az adatok megosztására:
  - Fájl alapú
  - Adatbázis alapú
  - Socket alapú

File alapú adatmegosztás

- Az adatok megosztásának első és legáltalánosabb módszere, a fájlok megosztása (elsőre túl egyszerűnek tűnhet)
- Minden operációs rendszer és hardver támogatja.
- Az egyik alkalmazás adatokat ír, a másik pedig adatokat olvas ugyanabból az állományból.
  - Ha mindkét alkalmazás egy gépen fut, akkor a merevlemezt mindketten használhatják.
  - Ha különböző gépeken futnak, akkor kell egy fájlátviteli módszer. (leggyakrabban (még mindig) FTP kapcsolat) FTP helyett SFTP, SCP szabványok alkalmazása szerencsésebb, mivel a kommunikáció ezeknél titkosított.
- A legtöbb file alapú megosztás szöveges állományokat használ.
  - Ennek oka az, hogy a számokat így a legkönnyebb különböző rendszerek között átvinni. A float vagy double számokhoz bináris állományok kellenének, amelyek már eltérő bytesorrendet is jelentenek különböző programozási nyelvek és operációs rendszerek között. Olyan is lehet, hogy egy integere egyik rendszeren 2 a másikon 4 byte-os.
- Legelterjedtebb a sima szöveges és XML alapú kommunikáció
  - Sima szöveg: fix hosszúságú rekordok és változó hosszúságú rekordok
  - Fix hosszúságúaknál az üres adatok esetén is ugyanannyi byte kerül átadásra, elválasztójelet nem kell alkalmazni a mezők elválasztására.
  - A változó hosszúságúaknál kell valamilyen elválasztójel az egyes rekordok elválasztásához . A legismertebb ilyen módszer a CSV (comma separated values)
  - XML alapú kommunikáció esetén az állomány XML, aminek előnye a szabványos forma, de hátránya a lassabb feldolgozás.
- Feldolgozás
  - Kiírásnál a számértékeket szöveggé kell átalakítani, és a beolvasásnál ugyancsak a szövegeket számmá kell átalakítani. Természetesen ehhez ismerni kell az állomány szintaktikáját.

File alapú adatmegosztás

- Ez a módszer máig is a legelterjedtebb, és a legtöbb hátránnyal rendelkezik
  - A megosztás nem valós idejű (real-time). Általában napi, heti, havi adatcseréhez érdemes használni. Ha a ciklusok között adatmódosítást végeznek, pl egy ügyfélnek megváltozik a címe, elképzelhető hogy a számlanyomtató alkalmazás még a régi címre küldi a számlát, mert csak később értesül a változásról.
  - Megbízhatatlan lehet ha nagyszámú állomány átvitele szükséges. (rsync azért segíthet!)
  - Mindkét alkalmazás fejlesztőjének (általában) ismernie kell:
    - az állomány formátumát
    - állományok elnevezési konvencióit
    - az állományok helyét
    - hogyan történik az állományok törlése?
    - lock-olási mechanizmust. Azaz ha az egyik alkalmazás ír az állományba, a másiknak nem szabad írnia!
    - Ha külön gépeken futnak az alkalmazások, akkor meg kell egyezni a fájl-átviteli módszerben

Adatbázis alapú megosztás

- Az adatmegosztás során az egyik alkalmazás nem állományba, hanem egy adatbázisba ír, a másik pedig olvassa az adatot.
  - Fő különbség a fájl alapúhoz képes, hogy az adatbázis az esetek többségében külön gépen üzemel. Valamint az adatok továbbítása még akkor is hálózaton keresztül történik, ha azonos gépen üzemel a DB. E miatt általában ez lassabb módszer mint a fájl alapú.
  - Előnye a módszernek, hogy a fájl alapúval ellentétben ez nem pont-pont kapcsolat. Azaz több alkalmazás is lehet író és olvasó.
  - Ez a módszer divatos az SQL alapú DB-k elterjedésével, mivel az SQL-t minden platform támogatja. (még a mobilok is, sqlite)

Adatbázis alapú megosztás

- Hátrányok:
  - Hasonlóan a fájl alapú adatmegosztásnál, ez a módszer sem valós idejű, mivel ha egy alkalmazás beír az adatbázisba, másik alkalmazás erről nem kap (automatikusan) értesítést.
    - Hogyan lehet mégis értesíteni a másik alkalmazást?
  - Az adatok tárolására nem alkalmazható általános adatformátum.
    - A kommunikációban résztvevő alkalmazásoknak Db specifikusnak kell lenniük
  - Több résztvevős kommunikáció esetében lockolási nehézségek léphetnek fel
    - Természetesen ezt könnyíteni lehet a DB motorok konfigurálásával, lásd MySQL innodb vagy myisam adatbázis alkalmazása, amikor az író és olvasó folyamatok nem csak tábla, hanem rekord szintű lockolást is megengednek.
  - Probléma lehet, ha nem LAN on hanem WAN-on kell kommunikálni

Socket alapú megosztás

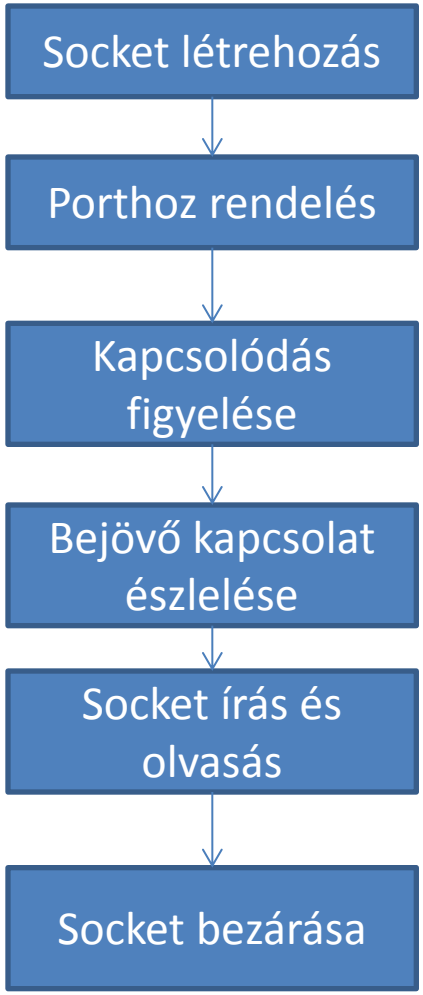
- Ha valós idejű megosztás szükséges érdemes socket alapú kommunikációt alkalmazni
  - A legegyszerűbb hálózati összekötést két gép között a socket-ek jelentik.
  - A socket nem más, mint egy kommunikációs végpont.
    - Azonos és különböző gépek közötti kommunikáció esetén is lehet socketet alkalmazni. Gondoljunk a Java VM-ek közötti kommunikációra.
    - A socketeket minden operációs rendszer támogatja. Manapság a HTML5 szabvány a websocket fogalmát is bevezeti!
    - A TCP/IP szabvány socketnek nevezi az IP címet és a hozzá tartozó portot.

Socket alapú megosztás

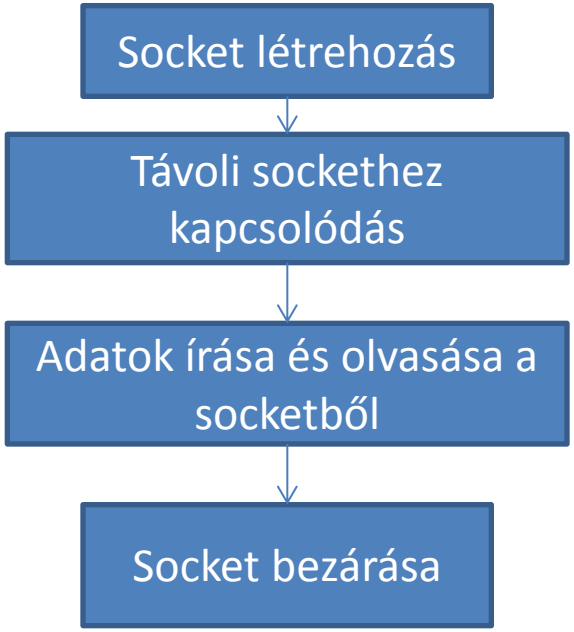
- Működés
  - ‚A' alkalmazás megnyit egy socketet, amin figyel. ‚B' alkalmazás rácsatlakozik a socketre és kommunikál ‚A'-val.
    - ‚A' alkalmazás rögtön reagál a bejövő adatokra, ahogy ‚B' írni kezd a socketen keresztül. E közben ‚C' alkalmazás is függetlenül csatlakozhat és írhat. Ezzel megvalósul a real time kommunikáció az alkalmazások között.
    - ‚A' alkalmazást fel kell készíteni a párhuzamos feldolgozásra. Ez a fejlesztő feladata: pl: Session Bean-ek esetén a fejlesztőnek kell felkészülnie a párhuzamos kérések helyes lekezelésére!
    - ‚A' alkalmazás szerver, ‚B' és ‚C' kliens szerepet tölt be a kommunikáció során.
- Modern kommunikációs rendszerek alapvetően socket alapon kommunikálnak a ‚rendszer mélyén'!

Socket alapú megosztás

Szerver oldal



Klens oldal



Socket alapú megosztás

- Hátrányai
  - Legfőbb hátránya az, hogy az adatok megosztását közvetlen módon támogatja, de a funkciók megosztását viszont nem
  - Socket API alacsony szintű interfész, aminek a programozása nem könnyű
  - Az alacsony szintű API miatt komplex adatok továbbítása nehézségekbe ütközik
  - Numerikus adatok továbbítása esetén a socket nem platfomfüggetlen
  - A kapcsolat típusa szoros, mivel valójában pont-pont kapcsolatot alakítunk ki
- Ezek ellenére mégis a socket alapú kommunikáció alapvető fontosságú
  - RPC és MOM rendszerek is alapvetően socket alapú kommunikációt használnak
- Ha az alkalmazások azonos gépen vannak akkor: shared memory, pipe és named pipe összeköttetést is lehet használni
  - Ezek a kapcsolatok nagyon gyors kommunikációt eredményeznek, de csakis 1 gépen futó alkalmazások kommunikációjára használhatóak!

RPC

- Új koncepciók:
  - Interfészek bevezetése
  - Szolgáltatást adó kiszolgáló (szerver) és a szolgáltatást igénybevevő (kliens) szemlélet itt jelenik meg először. A szerver szolgáltatja a szolgáltatások implementációját.
  - Marshalling koncepció a paraméterek átadására
  - Rendszer és hálózat specifikus funkciók egységbezárása (encapsulation) programkönyvtárként való alkalmazása
  - Kliens és szerver stub-ok bevezetése
  - Platformfüggetlenség XDR (external data representation) bevezetése

- A szinkron RPC nek történetileg 2 változata alakult ki
  - Restricted RPC -> nincs valódi hálózati kommunikáció, operációs rendszer függő, Solaris implementáció
  - RCP – van hálózati kommunikáció

Microsoft specifikus komponensek

- OLE (Object Linking and Embedding) technológia (1991) → 1992-ben a Windows 3.1 alapfunkciója lesz, alkalmas arra hogy egy MS Excell táblát egy MS Word dokumentumba ágyazzuk.
- 1991 Microsoft bemutatja a Visual Basic vezérlőket (VBX)
- 1993-ban megjelenik az OLE2 (alatta pedig a COM objektummodell), ami már nem „egyszerű” beágyazás, hanem egy vállalti szintű komponenstechnológia.
  - OLE vezérlők OCX nevet kapnak
- 1996, az OLE internettel kapcsolatos részei ActiveX elnevezést kaptak, majd bemutatták a DCOM technológiát a CORBA ellensúlyozására.
- COM+ megjelenik a Windows 2000-el.
  - Tranzakciókezelés (Microsoft Tranzaction Serverrel - MTS)
  - DCOM ból átveszik az elosztott komponensek használatát.
- Az Activex vezérlők Internet Explorerbe ágyazhatósága elősegítette a vírusok gyors terjedését. (1996)

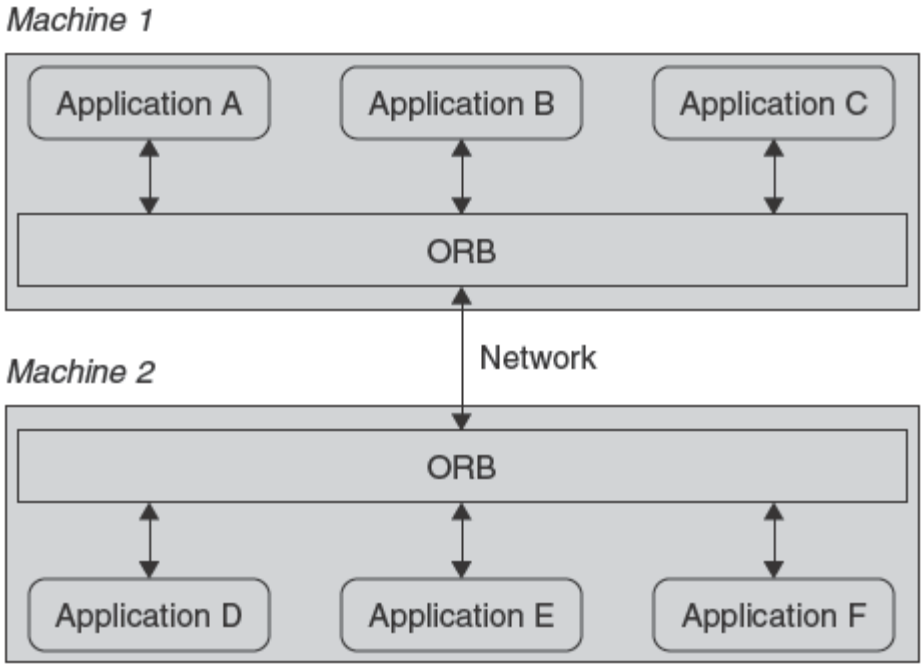
Microsoft specifikus komponensek

- A COM komponensek (class ID, CLSID) GUID-ok alapján azonosíthatóak. (globális egyéni azonosító 16 byte hosszú adat)
- COM komponensek interfészei (interface ID, IID) is GUID-ok
- A COM objektumokhoz interfészeiken keresztül lehet kapcsolódni. Minden interfész az IUnknown interfészből származik, a következő 3 metódust implementálni kell:
  - AddRef() – referencia számlálás
  - Release() – élettartam vége
  - QueryInterface() – mutatókat ad vissza a komponens által megvalósított interfészekre
- IDL és Típus könyvtárak- Interface Definition Language
  - MIDL compiler segítségével „compiler független header” fájl hozható létre, valamint C++ proxy forráskód.
  - MIDL compiler típus könyvtárakat is fordít (TLB) - bináris metadata nyelv független COM típusokhoz

Microsoft specifikus komponensek

- Registry tárolja a GUID-okat:
  - *HKEY_CLASSES_ROOT\CLSID - osztályok*
  - *HKEY_CLASSES_ROOT\Interface – interfészek*
- Windows XP bevezette a „RegFree COM”-ot, ahol a metadata XML-ben tárolódik (assembly manifest), az exe-be ágyazva akár, vagy külön. Ez megengedi, több komponens verzió párhuzamos elérését is. Az alkalmazás indításakor a rendszer megkeresi a megfelelő manifest filet. Ha nem található, akkor a registry alapján folytatja a keresést.
- Lehetséges problémák:
  - Referenciák feloldása – körkörös hivatkozások
  - DLL Hell.

Corba (ORB – Object Request Broker)

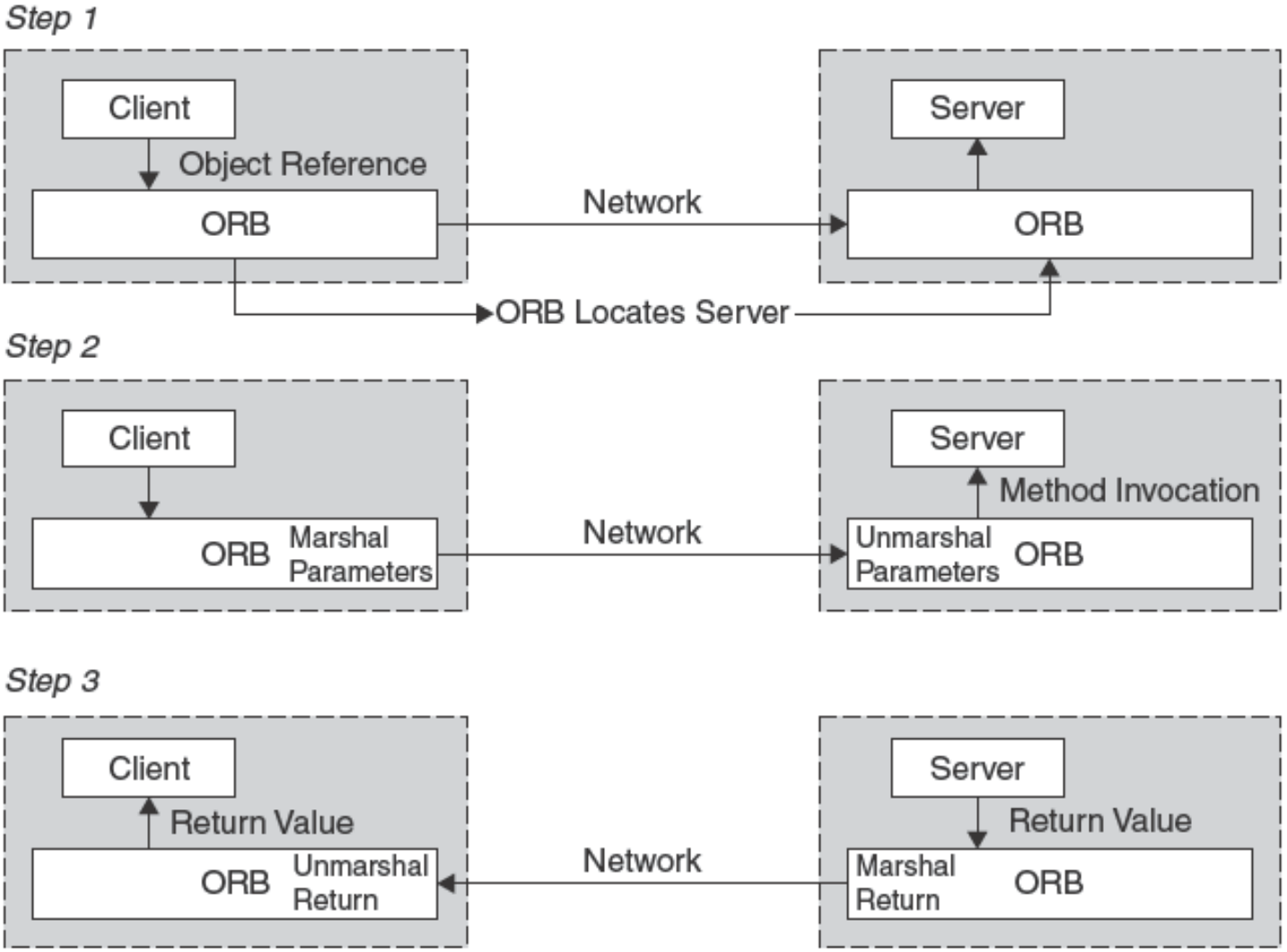


- A Corba koncepció a különböző alkalmazások és platformok egyszerűbb integrációját azzal valósítja meg, hogy az egyes gépeken az ORB komponensek kommunikálnak közvetlenül az alkalmazásokkal, a gépek közötti kommunikáció az ORB-k között zajlik. Ez a koncepció hasonlít a későbbi ESB megoldásokhoz is.
- 1991-ben jelent meg az első vátozata

Corba Model

- A Corba koncepció másik fontos tulajdonsága, hogy definiál egy platform független interface definition language (IDL)-t. Az interfész koncepció már megjelenik az RPC-nél is, de itt az IDL nyelv független lehetőségeket biztosít. (C++, C, Java, COBOL, Smalltalk)
- IIOP – Internet Interface ORB protokoll segítségével kommunikál. TCP/IP alapú, ami ma már természetes, de akkoriban ez újdonság volt. A kommunikációt kliens stub ok és szerver skeleton-ok segítik

Corba Kommunikáció



Corba IDL

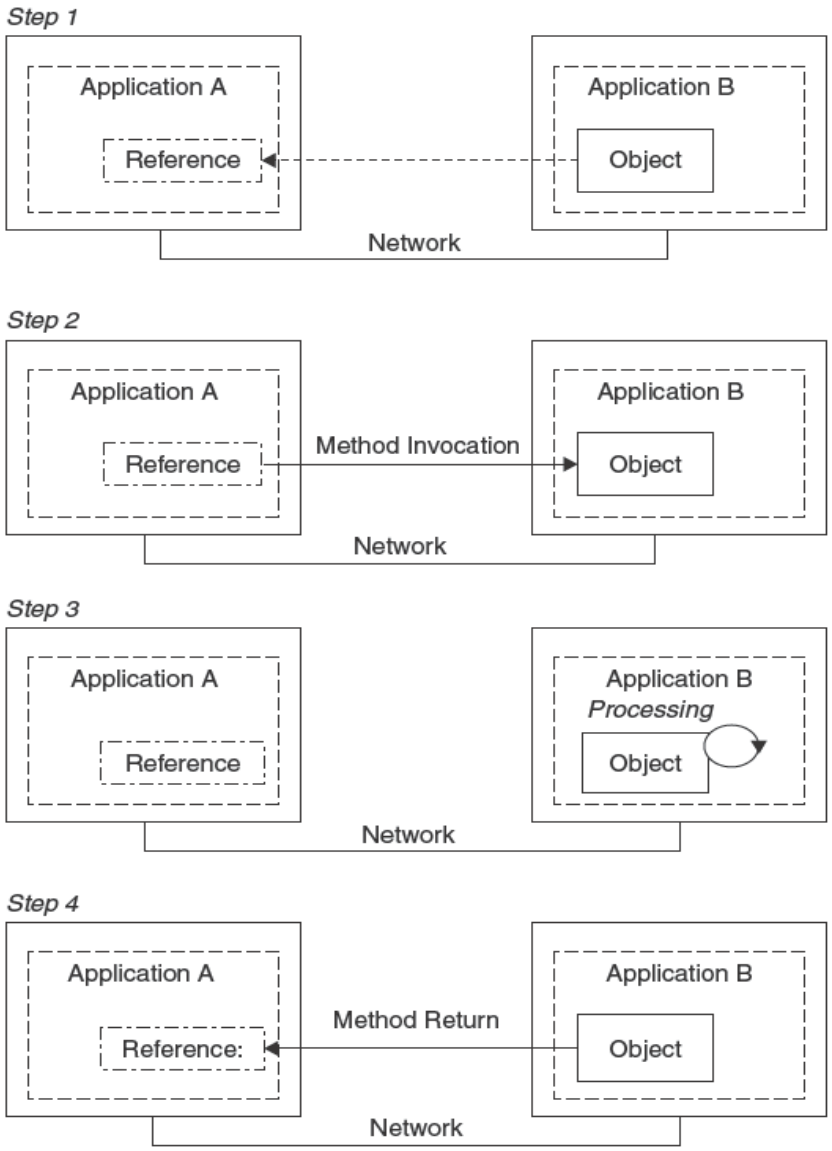
- Interfészek IDL szintaktikája a következő példa alapján jól látható:

```
module Test {
  interface square {
    attribute double arg1;
    double getSquare (in double
                      arg1);
    };
  };
```

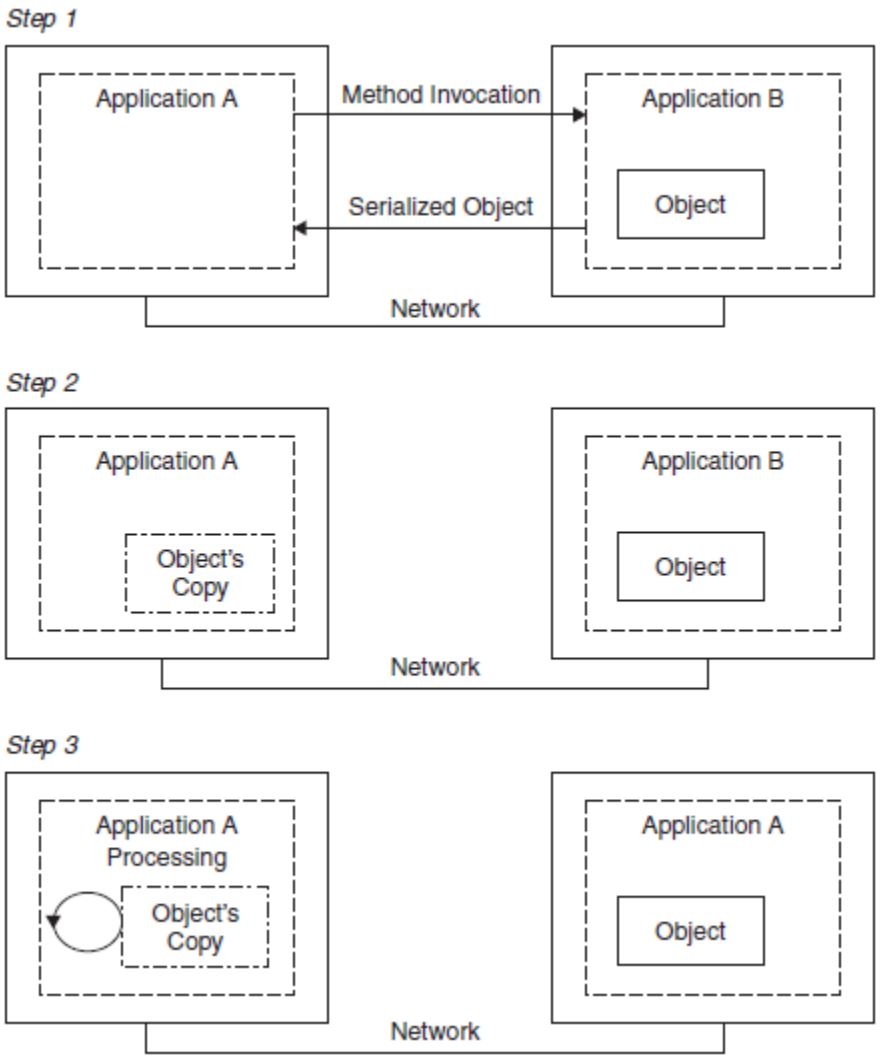
- Az IDL alapján automata eszközök generálják a megfelelő kódot, ami a gyakorlatban több fájlt jelent. A különböző nyelvekhez különböző generátorokat fejlesztettek: pl. Java nyelvhez a „idl2j” generátor szolgáltatja a kliens és szerver stub-okat.

Corba Objektum modell

- A Corba modell 3 fő elemet tartalmaz
  - Közel transzparens objektum elosztás támogatása
  - Objektum referenciák
  - Objektum adapterek (ez biztosítja az elosztott objektumok egymás közötti kommunikációját)
- Transzparensnek nevezzük az objektumok elhelyezkedését, mivel a kliensnek nem kell tudnia hol helyezkedik el a távoli objektum. A távoli objektumok használata megegyezik a lokális objektumok használatával. A rendszer elosztott természetét a Corba modell biztosítja.



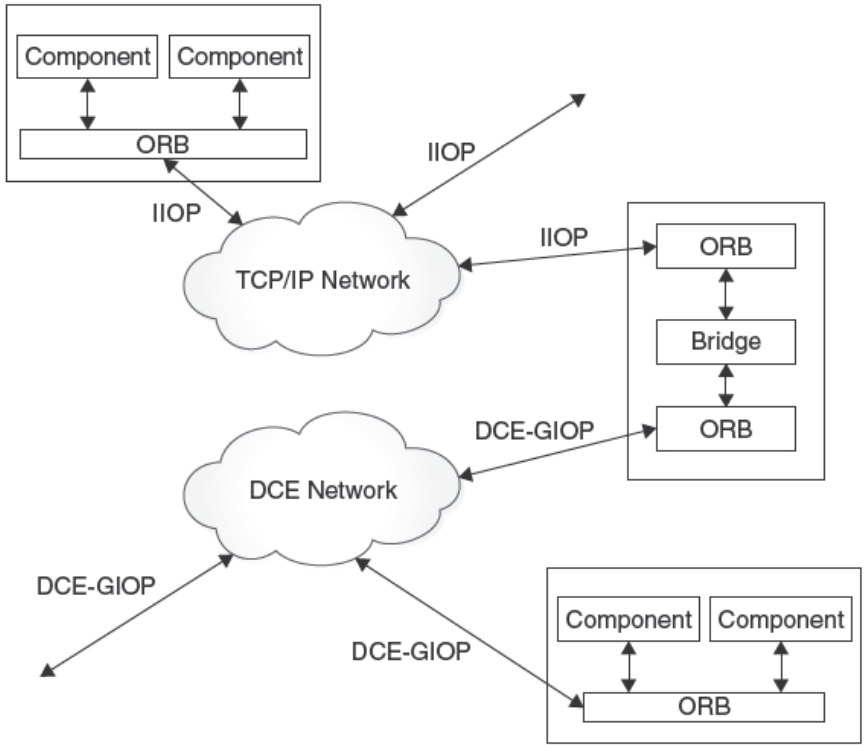
Referencia alapú kommunikáció: a távoli objektumon hajtódik végre a művelet



Érték alapú kommunikáció: az objektum átkerül kliens oldalra!

Corba Bridge

- Különböző típusú hálózatok közötti kommunikációt a Corba Bridge teszi lehetővé, ami közvetlenül nem érhető el.



Corba szolgáltatások

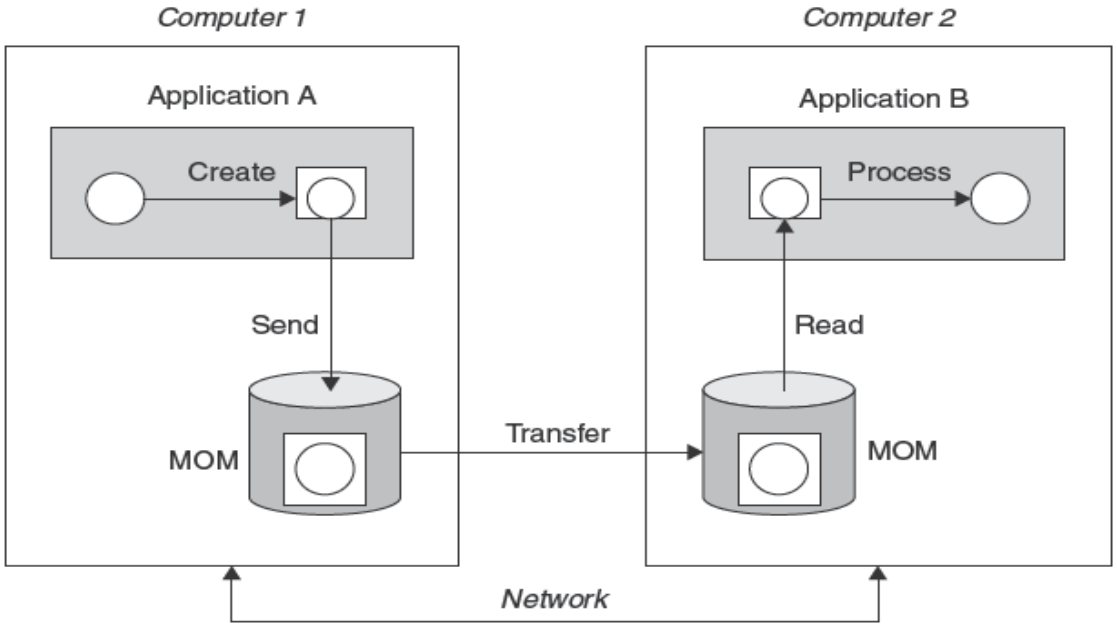
- Object Management Architektúra (OMA)
  - Névszolgáltatás: corba objektumokat regisztrálni kell a névszolgáltatóba és név alapján érhetőek el. (SOA megoldás elődjének tekinthető)
  - Biztonsági szolgáltatás: elosztott rendszerekben definiálni kell, hogy melyik erőforrás érhető el mások számára.
    - Authentication: megállapítja, hogy a felhasználó az-e akinek mondja magát
    - Authorization: megállapítja hogy az authentikált felhasználó hozzáférhet-e az adott szolgáltatáshoz
    - Security auditing: minden felhasználói tevékenység eltárolható
    - Nonrepudiation: digitális aláíráshoz hasonlóan biztosítja, hogy az objektum/adat nem változtatható meg.
  - Konkurencia vezérlő szolgáltatás: különböző típusú lock-olási eljárásokat alkalmaz a stabil konkurencia biztosításához.
  - Tranzakciós szolgáltatás: elosztott rendszerekben célszerű a tranzakciókat központilag kezelni. Pénzátutalásnál fontos, hogy minden résztvevő kapjon commit-ot, vagy rollback-et hiba esetén
  - Életciklus szolgáltatás: Corba objektumok létrehozása-, törlése-, másolása-, mozgatásáért felelős.

Üzenetkezelés (messaging)

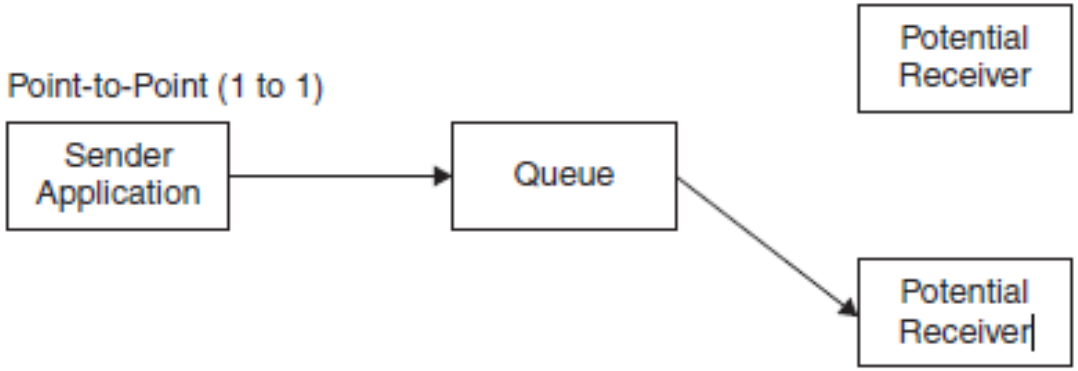
- RPC, ORB alapú kommunikációban nem lehet könnyen megvalósítani a garantált kézbesítést
  - Ha a szolgáltató nem érhet el, a kliens várakozik, vagy hibával leáll.
- Üzenetek küldése mindig aszinkron, azaz a kliens nem vár visszatérő értéket a kiszolgálótól.
  - Az alkalmazások ezért nem is közvetlenül kommunikálnak, hanem üzenetsorok segítségével
  - Marshalling, unmarshalling mechanizmusra itt is szükség van

Üzenetkezelés (messaging)

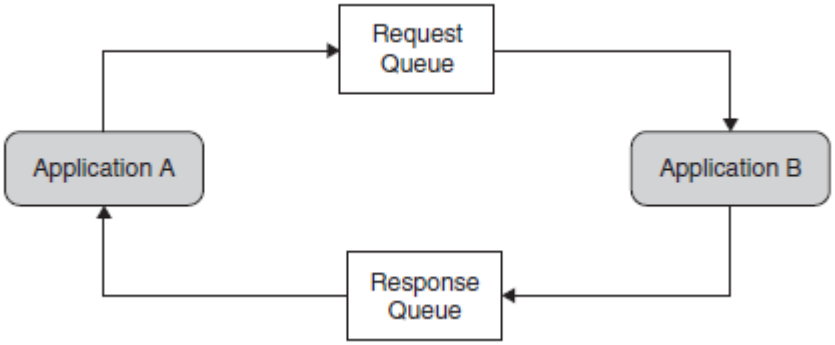
1. Az A alkalmazás létrehozza az üzenetet
2. Az A alkalmazás elküldi a MOM-nak az üzenetet
3. Az üzenetkezelő átküldi az üzenetet
4. A B alkalmazás olvas az üzenetsorból, majd feldolgozza az eredményt



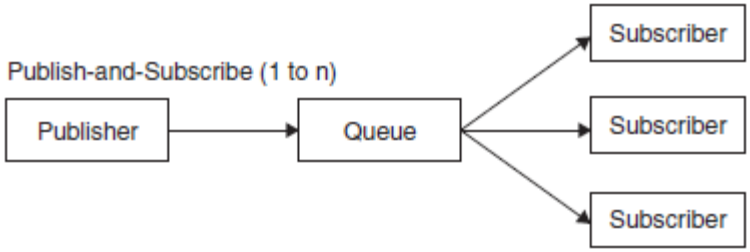
- Pont-Pont csatorna



  - Szinkron üzenetküldés szimulációja pont-pont csatornával



- Publish-subscribe üzenetkezelés



A feliratkozók mindegyike megkapja az üzenetet

Az üzenet fejléce a következő információt tartalmazhatja:

- Üzenetazonosító (id) /korrelációs id
  - Segítségével a kliens később hivatkozhat egy adott üzenetre
- Persistent/nonpersistent
- Visszatérési cím
  - Az esetleges válaszüzenetek hová legyenek továbbítva?
- Prioritás
  - Nagyobb prioritású üzenetek hamarabb lesznek továbbítva (pl: az rtmp protokoll is ilyen)
- Szegmentációs/ csoport információ
  - Ez a funkció rejtve marad a programozótól. Ha egy üzenet nagy méretű akkor nem lassítja a rendszert azzal, hogy egyben küldi át az üzenetsorra, hanem darabolja az üzenetet a rendszer
- Időbélyeg
- Üzenet élettartam (meddig legyen az üzenet érvényes?)
  - Ha az élettartam lejár, az üzenet nem törlődik automatikusan, nincs háttérfolyamat, amely időnként ellenőrzi, hogy az üzenet érvényes-e vagy nem. Az üzenet addig marad a soron fizikailag, ameddig valamelyik kliens (feliratkozó) nem töltené le, de természetesen ekkor nem fog letöltődni.
- Verzió

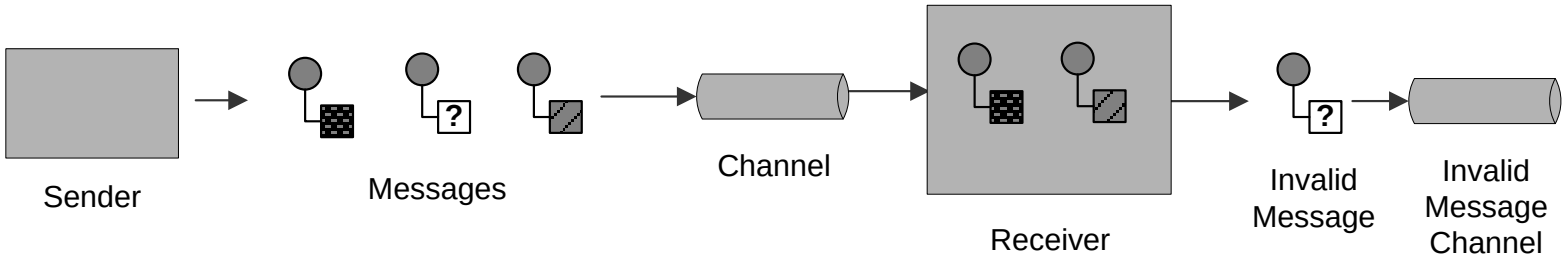
- JMS üzenettípusai
  - TextMessage
    - Egyszerű string üzenet, ami lehet természetesen XML
  - BytesMessage
    - Bináris üzenetek továbbítása
  - ObjectMessage
    - Java Object-ek továbbítása
  - StreamMessage
    - Primitív java adattípusok adatfolyama: int, char, long, float
  - MapMessage
    - Kulcs/Érték párok java.util.Map hoz hasonlóan

Üzenet vezérelt Bean

- Message-driven beans (MDBs)
  - Állapotmentes, szerver oldali J2EE komponensek
  - Előnye: Az üzenetfogadás és feldolgozás párhuzamos. MDB container menedzseli a párhuzamosságot, így a programozó az üzleti logikára koncentrálhat. Több száz üzenetet is kaphat párhuzamosan, a container több példányt is indíthat párhuzamosan a bean-ből.

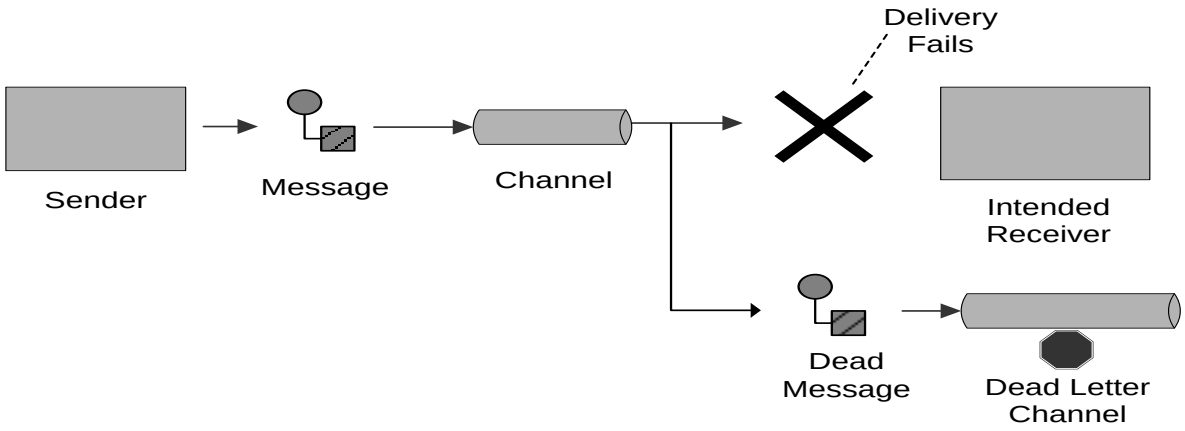
Érvénytelen üzenet csatorna

- Minden rendszer fejlesztésekor problémát jelent az, hogy a működő rendszer nagyon zavarja a fejlesztői munkát. Egy hiba bekövetkezése nem naplózható ki a lokális fájlrendszerbe, mert egy nagy rendszer esetén ez akár több száz fájlt is jelenthet, mindegyiket végignézni csak azért, hogy megtudjuk, hogy hol keletkezett a hiba nem hatékony megoldás. Ezért szokás hibajelentő csatornákat létrehozni, amelyek segítenek összegyűjteni a hibákat.
- Az ilyen hiba-gyűjtő csatornák egyike az érvénytelen üzenet csatorna, amely kifejezetten az üzenetek feldolgozási hibáit hivatott gyűjteni. Ha egy alkalmazás nem tud feldolgozni egy üzenetet, mert az formailag hibás, vagy mert az üzenetben foglalt feladat teljesíthetetlen, akkor erről a problémáról jelentést kell tennie.
- Az érvénytelen üzenetek csatornáján normál működés esetén nem lehet üzenet, azonban az alkalmazások rejtett hibái, vagy a fejlesztés során elkövetett hibás módosítások miatt megjelenhetnek üzenetek.
- Nagyon gyakran előre gyártott hibaüzenet feldolgozó egységet is tartalmaz, amely értesíti a rendszer üzemeltetőit a hibáról (például elektronikus levélben).



Halott levél csatorna

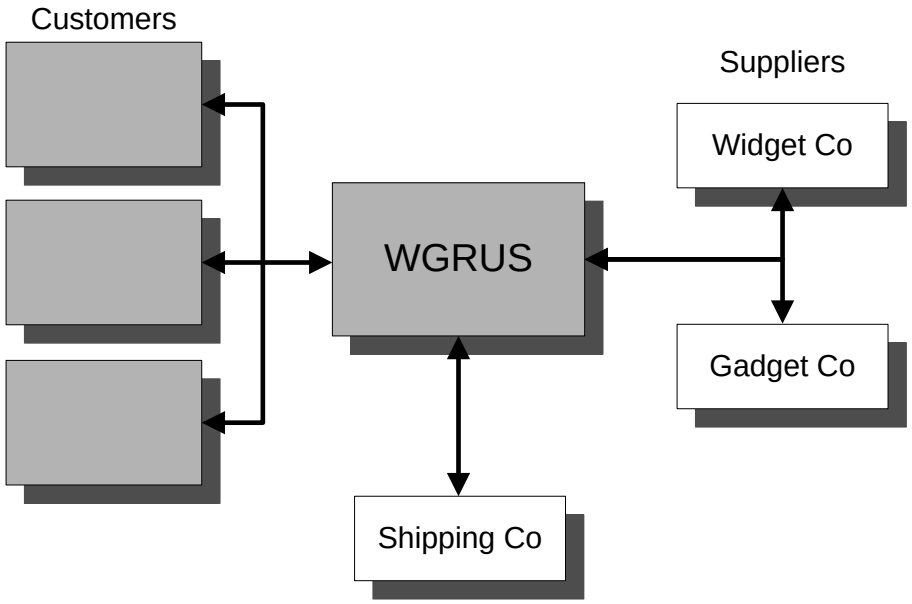
- Az üzenetküldő rendszerek szintén beépítve tartalmazzák, elsősorban az üzenetküldő rendszer küldhet üzenetet rajta, a rendszer karbantartói számára.
- A halott levél vagy olyan formai hibát tartalmazó üzenet, ami miatt kézbesíthetetlen (például érvénytelen címzett), vagy a csatornát, amiben az egyébként érvényes üzenet haladt megszüntették, vagy az üzenet élettartama lejárt a kézbesítés előtt.
- A halott levél és az érvénytelen üzenet közötti legfontosabb különbség az, hogy a halott levél valamiért nem kézbesült, ami akár még az üzenetküldő rendszer hibás működését is jelentheti, ellentétben az érvénytelen üzenettel, ami sikeresen megérkezett a címzetthez, de a címzett alkalmazás nem tudta feldolgozni azt.
- Ezek az üzenetek mindig komoly hibára utalnak a rendszer helyes működéséhez képest, ezért minden üzenetküldő rendszer nagy hangsúlyt fektet a kezelésükre.



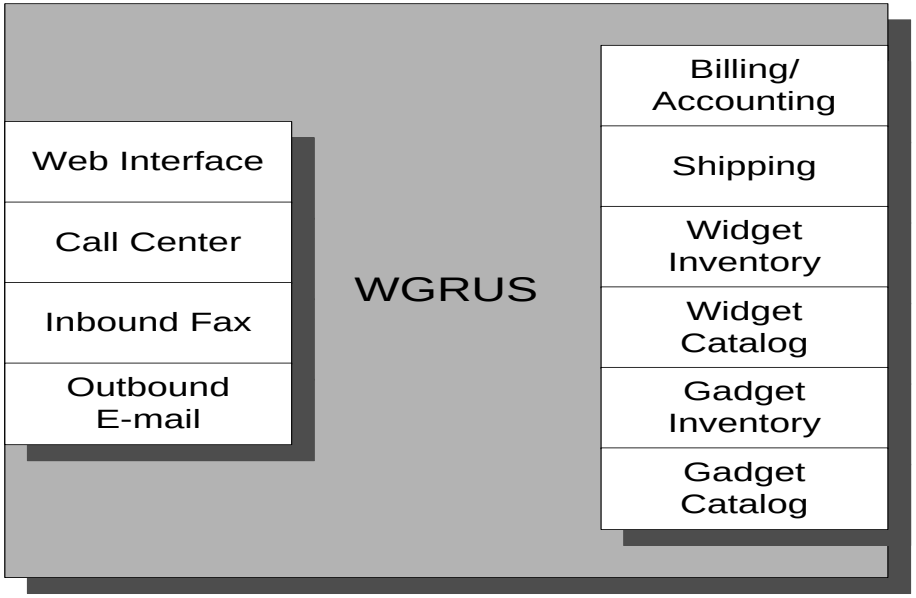
A „WGRUS” példa-vállalat

- A WGRUS vállalat egy kereskedelmi vállalat, kezdetben egy beszállító termékeit forgalmazta, de a későbbiekben ezt bővítette egy másik beszállító termékeivel. A két beszállító „Widget” – eket és „Gadget” eket gyárt, innen a neve:

  Widgets & Gadgets’R Us (WGRUS).



WGRUS belső szerkezete



A vállalat kapcsolattartó egységei:

- Egy Internetes portál, és a mögötte lévő informatikai osztály,
- Egy telefon központ, amely egy készen vásárol szoftver segítségével adminisztrálja a bejövő kéréseket.
- Fax rendszer. A faxon érkező kéréseket részben kézzel, részben egy házilag épített programmal veszik fel az alkalmazottak.
- A vállalat az ügyfelei felé e-mailben kommunikál.

A vállalat belső osztályai:

- Számlázás, egyenleg-kezelés
- Szállítmányozás
- Widget raktár (hagyományos okokból, a két beszállító kezelése egymástól független)
- Widget katalógus
- Gadget raktár
- Gadget katalógus

Elvárt eredmények

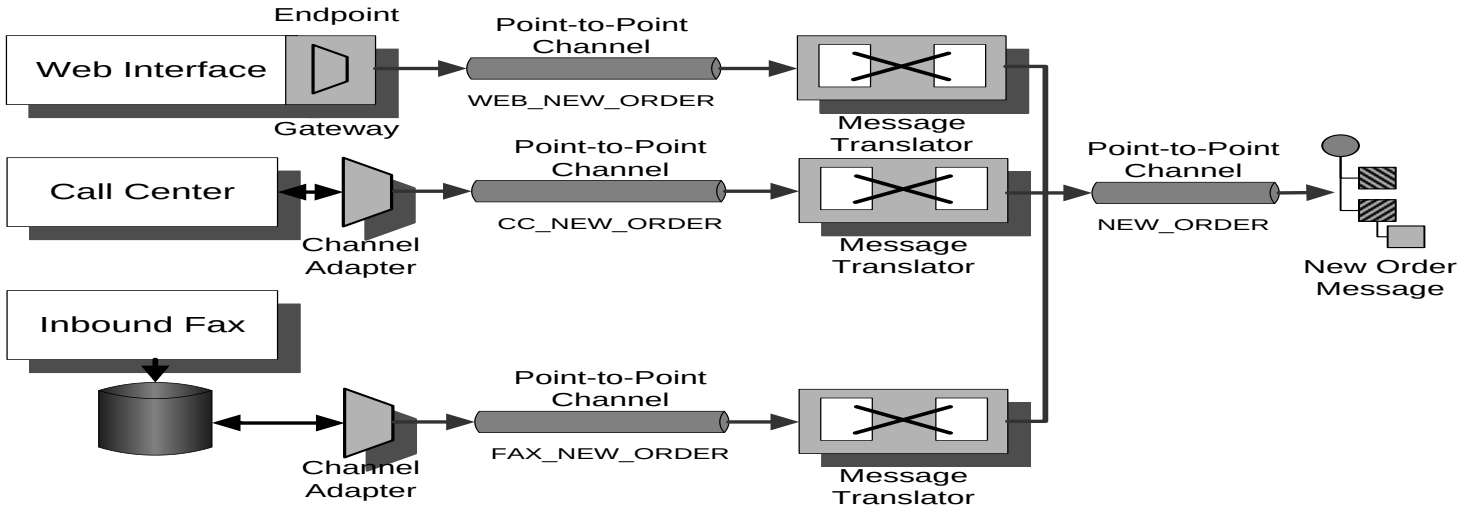
A vállalat elvárásai a kialakítandó rendszer felé:

- A megrendelések mindhárom irányból (Internet, telefon, fax) érkezhessenek be,
- A vevők egységes felületen keresztül láthassák a teljes kínálatot, függetlenül attól, hogy melyik beszállítótól érkezik,
- Az ügyfelek az adataikat a Webes interfészen keresztül bármikor módosíthassák,
- A Webes interfészen a megrendeléseik állapotát nyomon követhessék,
- A vállalat belső működését ne veszélyeztethesse a többi vállalat szoftvere.
- Az integráció megvalósítása ne veszélyeztesse a meglévő rendszerek stabilitását,
- Az integráció megvalósításakor a lehető legolcsóbb, és leggyorsabb megoldásokat kell választani.

A megvalósítás

- A megvalósítás első lépése a megrendelések fogadása, a megrendelések egységes alakra hozása, a forrástól függetlenül. Ennek elérése érdekében mindhárom rendelés felvevő eszközt alkalmassá kell tenni egy közös formátumú megrendelés objektum létrehozására.
- Mivel a megrendelések fogadása egy aszinkron művelet, és sok rendszert összekapcsol, ezért üzenetküldő rendszer kialakítását választjuk.
- Mivel a Web interfész saját fejlesztésű alkalmazás, ezért ezt bővíthetjük egy üzenetküldő rendszer végponttal, amely így közvetlenül képes kommunikálni a rendszerrel. A telefonközpont szoftveréhez egy modult tudunk fejleszteni, amely kommunikál a rendszerrel, míg a fax üzenetek kezelésénél csak az adatok adatbázisból történő „kilopásával” tudunk kapcsolódni a rendszerhez.
- Mivel mindhárom eszköznek saját adatformátuma van, ezért ezeket egységes alakra kell hozni, mivel a rendszer többi része felé transzparensé szeretnénk tenni az üzenet forrását. Ez azt eredményezi, hogy átalakítókat kell használnunk, minden üzenet-típusra.

A következő ábra a megrendelések fogadásának elvi rajzát mutatja:



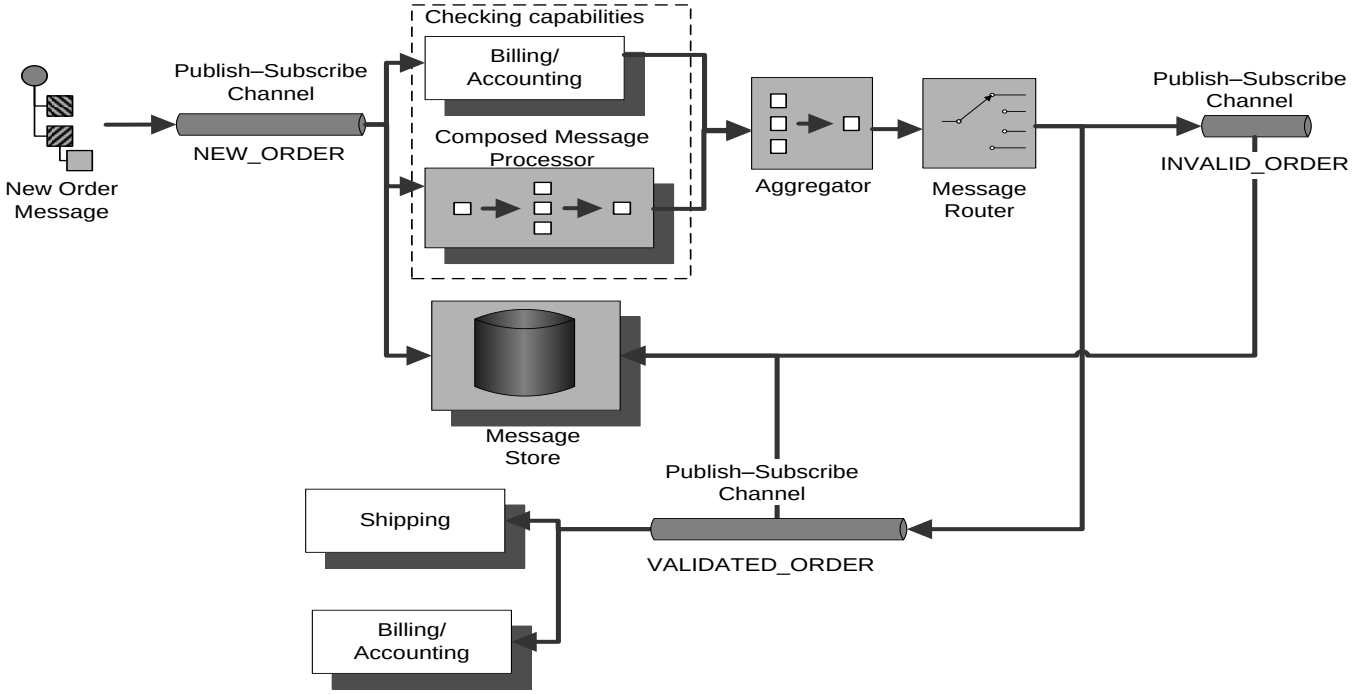
- Mivel a megrendelések fogadása egy aszinkron művelet, és sok rendszert összekapcsol, ezért üzenetküldő rendszer kialakítását választjuk.
- A megvalósítás során az egyes fogadó egységeket az előző ábrán látható módon alkalmassá tesszük az üzenetküldő rendszerrel való kommunikációra, majd az egyes üzenet típusokat egy-egy fordító segítségével közös formátumra hozzuk.
- Ez a közös formátumú megrendelés a NEW_ORDER nevű csatornán lesz elérhető a rendszer többi eszközei számára.

A bejövő kérések kiszolgálása

A megrendelések teljesítése négy fő műveletet jelent:

- A megrendelő számlaegyenlegének ellenőrzése (képes e kifizetni a megrendelését),
- A raktárakban megvan e minden megrendelt termék (képesek vagyunk-e szállítani),
- A szállítást meg kell rendelni a szállítmányozó cégtől,
- Számla kiküldése a megrendelőnek.

Mivel az ellenőrzések közül bármelyik sikertelensége meg kell, hogy akadályozza a további műveleteket, ezért két fő csoportban hajtjuk végre a feladatokat. Ha minden ellenőrzés sikeres volt, akkor szállítunk, és számlázunk. Az alábbi ábra részletesen bemutatja a folyamatot.

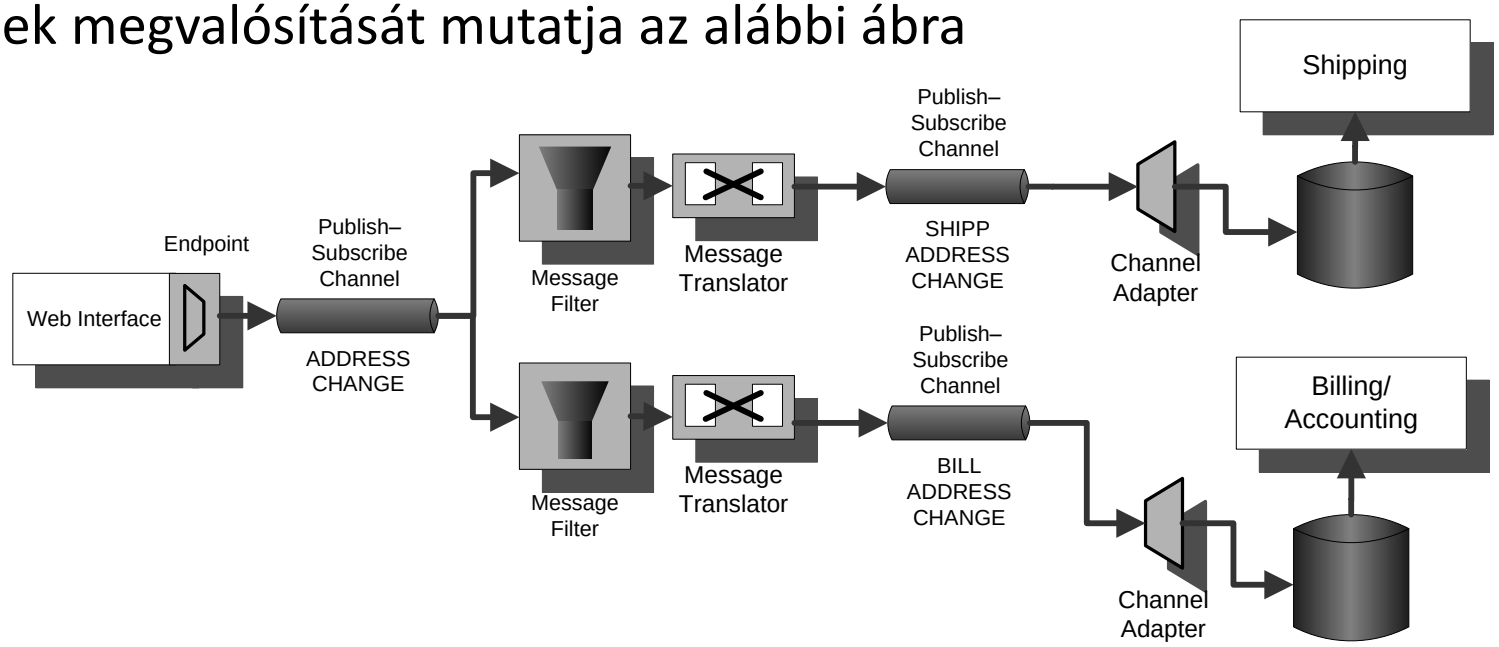


- Érdemes megjegyezni, hogy a megrendelés több termékre vonatkozik, melyek a vállalat két raktárának tartalmából tevődik össze. Mivel a raktáraknak külön kezelő szoftvere van, ezért a bejövő összetett megrendelést szét kell szedni elemeire, majd szűrők segítségével az egyes raktárak felé irányítani őket. Ezek után az egyedi, ellenőrzött megrendelési kéréseket egy aggregátor segítségével újra össze kell főzni egy elemmé.
- Az ilyen „szétszedés-művelet-összeillesztés” folyamatok nagyon gyakoriak, és egységesen Composed Message Processor-nak nevezzűk őket.

A karbantartási műveletek

- A rendszer karbantartása alatt minden olyan műveletet értünk, ami a rendszer folyamatos üzemének fenntartását biztosítja. Így ide tartozik, a legnehezebben kezelhető karbantartás is: a felhasználó adatainak kezelése. Természetesen ezt a folyamatot is automatizálni kell, erre egy lehetséges példa a felhasználó cím módosítási kéréseit kezeli.
- A címmódosítás kétféle cím módosítását jelentheti, a számlázási cím megváltozását, vagy a szállítási cím módosítását. Természetesen itt is meg kell hagyni a lehetőséget a felhasználónak, hogy egyszerre mindkét címet módosíthassa, és itt is egységes felületet kell biztosítani.

Ennek megvalósítását mutatja az alábbi ábra

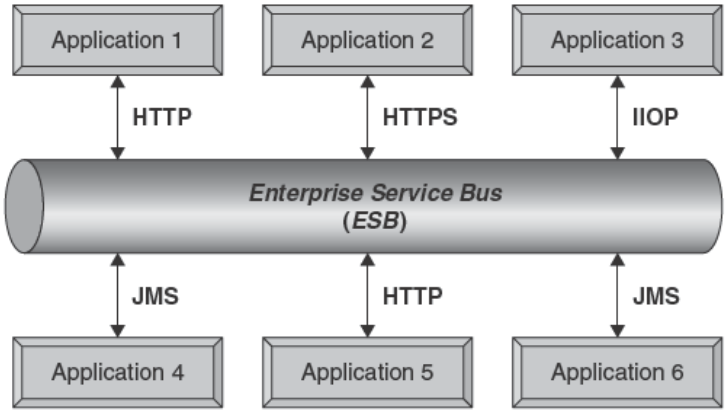


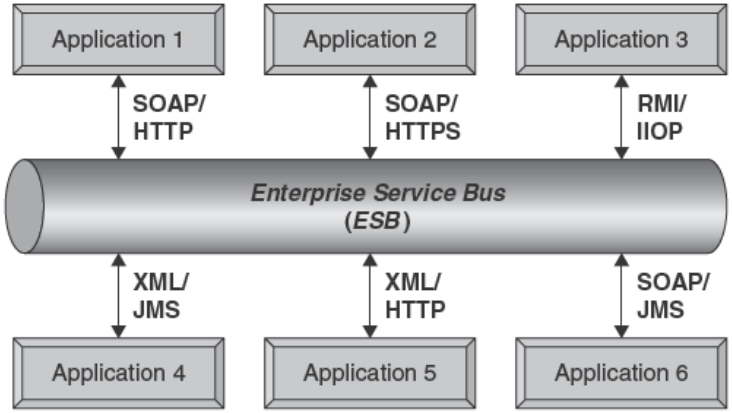
ESB – Enterprise Service Bus

- N darab alkalmazás integrációjánál a pont-pont kapcsolatok száma: N(N-1)/2. Azaz pl. 10 alkalmazás esetén 45 féle kapcsolatot jelent.
- Busz alkalmazása esetén N darab kapcsolat elegendő.
  - Új alkalmazás integrációja esetén a korábbiakon nem kell változtatni.
  - A szolgáltatási végpontokat a busz ismeri, az alkalmazásnak a busz címét elég ismernie.
  - A kezdeti buszok az ORB és üzenetsor funkciókat valósították meg.

Alapfunkciók:

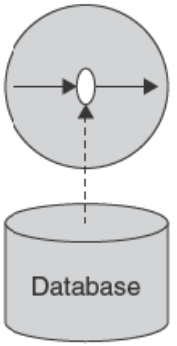
- Context/content based routing
- Protokoll transzformáció
- Adat/üzenet transzformáció



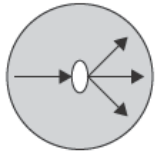


- ESB virtualizációs eljárások
  - Location and identity virtualization
    - service provider – service consumer (elhelyezkedésük rejtve marad egymástól), ebből az is következik, hogy kicserélésükkor a rendszert nem feltétlenül kell leállítani.
  - Interaction protocol
    - Service provider – service consumer (nem kell ismerniük egymás adatcsere formátumát és hálózati protokollját)
  - Interface
    - Service provider – service consumer (nem kell megegyezniük semmilyen interfészben
- Nem funkcionális követelmények – QoS
  - Teljesítmény és megbízhatóság
    - Pl: 50 milliszekundumon belül meg kell érkeznie az üzenetnek
    - Pl: 99.999% ban működnie kell a szolgáltatásnak
  - Biztonsági szolgáltatások
    - A biztonság fontos az elosztott rendszerekben, főleg akkor ha egy harmadik féltől származó komponenseket is alkalmazunk.
      - Adattitkosítás
      - Adatok integritásának megőrzése (ne lehessen módosítani az adatokat illetékteleneknek)
      - Auditing szolgáltatás (audit trail) a felhasználók tevékenysége rögzítésre kerül

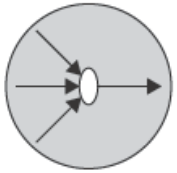
Opcionális ESB tulajdonságok



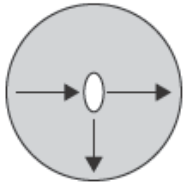
- Data enrichment
  - Az üzenet kiegészítése adatokkal. Pl az irányítószámhoz a várost hozzárendeljük
- Elosztás
  - Consumer alkalmazások feliratkoznak egy adott típusú üzenetre és az ESB elosztja az üzeneteket közöttük

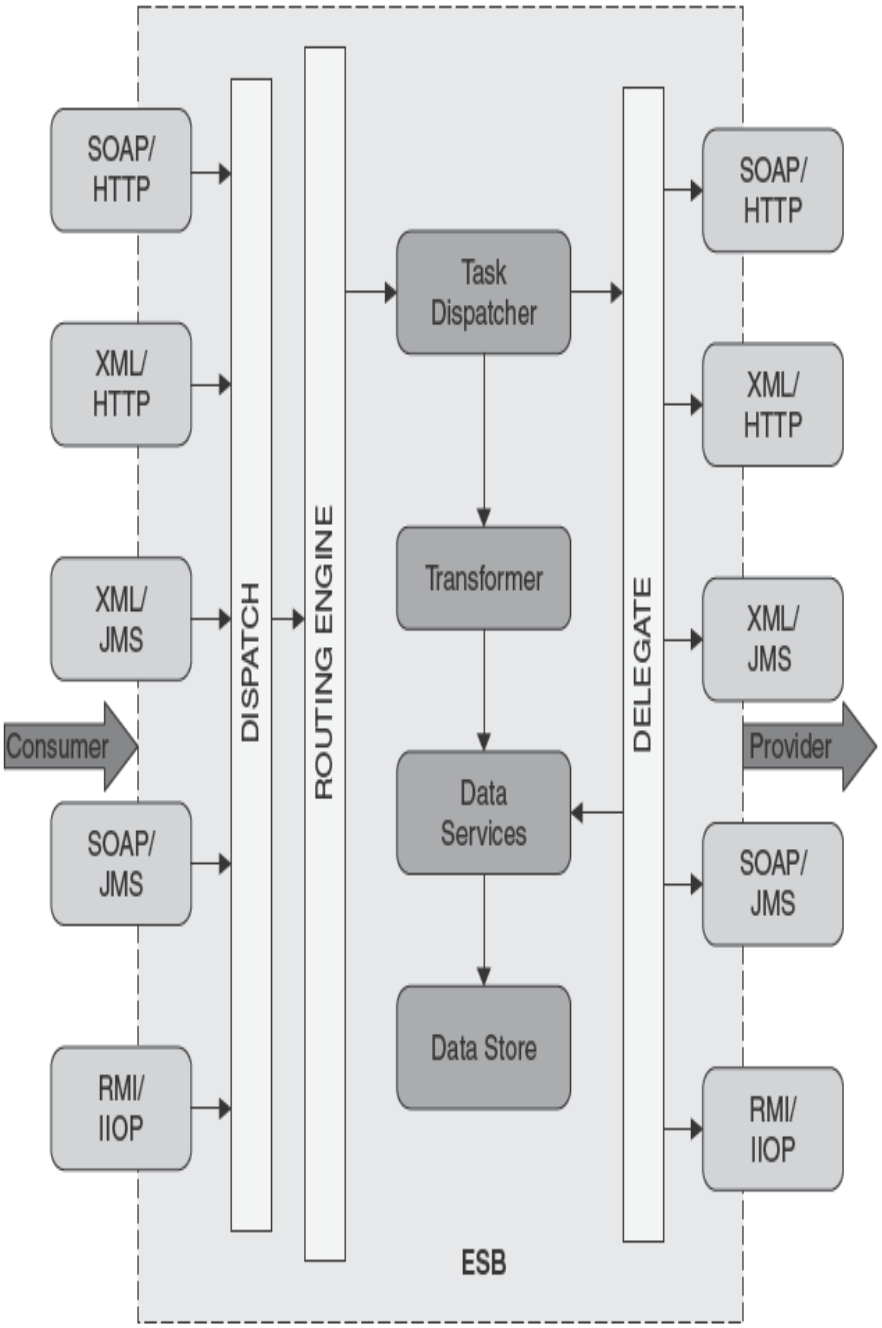


- Korreláció
  - Több üzenetből vagy esemény stream-ekből egy komplex esemény jön létre. Ez szabályalapú vagy minta felismerő technikákon alapszik.



- Monitorozás
  - Loggolás, adott feltételeknek megfelelő üzenetek szűrése

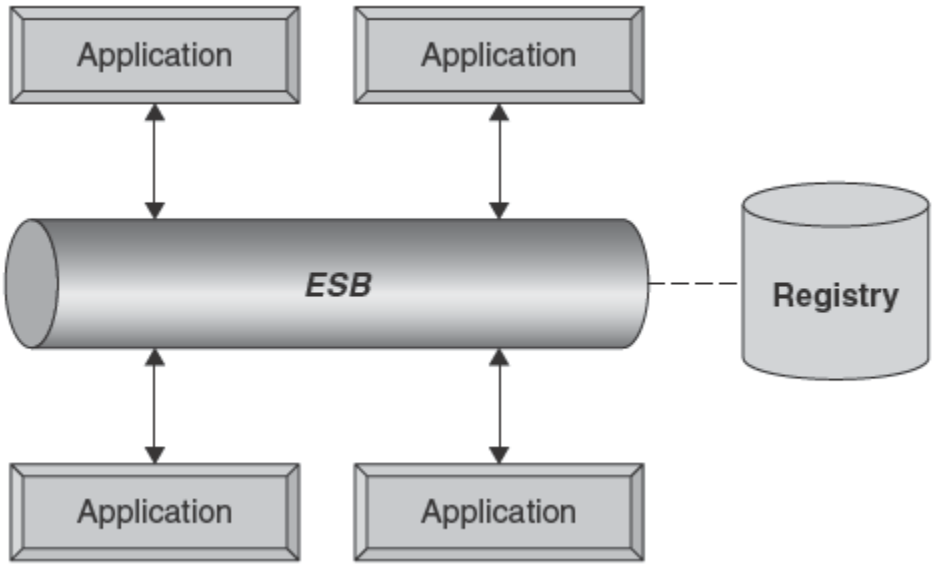




- Adapterek

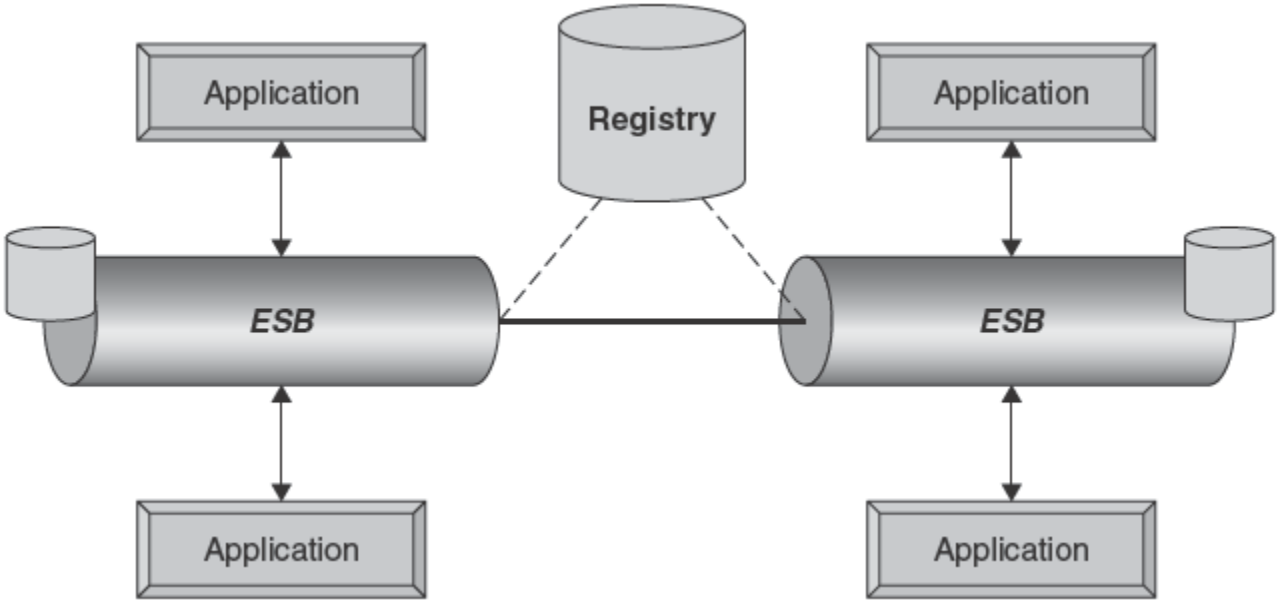
  Minden kimenő/bemenő adat adaptereken keresztül áramlik (SOAP/HTTP,XML/HTTP stb..)
- Dispatcher
  - Központosított belépési pont
  - Kontextus alapú routing
- Routing and rules engine
  - A transzformációk (adat és data enrichment) végrehajtásáért felelős
- Service delegates
  - Végpontokat vezéri
- Loggolás, kivételkezelés

- Telepítési konfigurációk
  - Kisebb rendszereknél elég 1 ESB és egy registry. → golbal ESB-nek nevezzük.
  - Nagyobb rendszerek esetén több registry szükséges, ESB-k ilyenkor gateway-ként is funkcionálnak

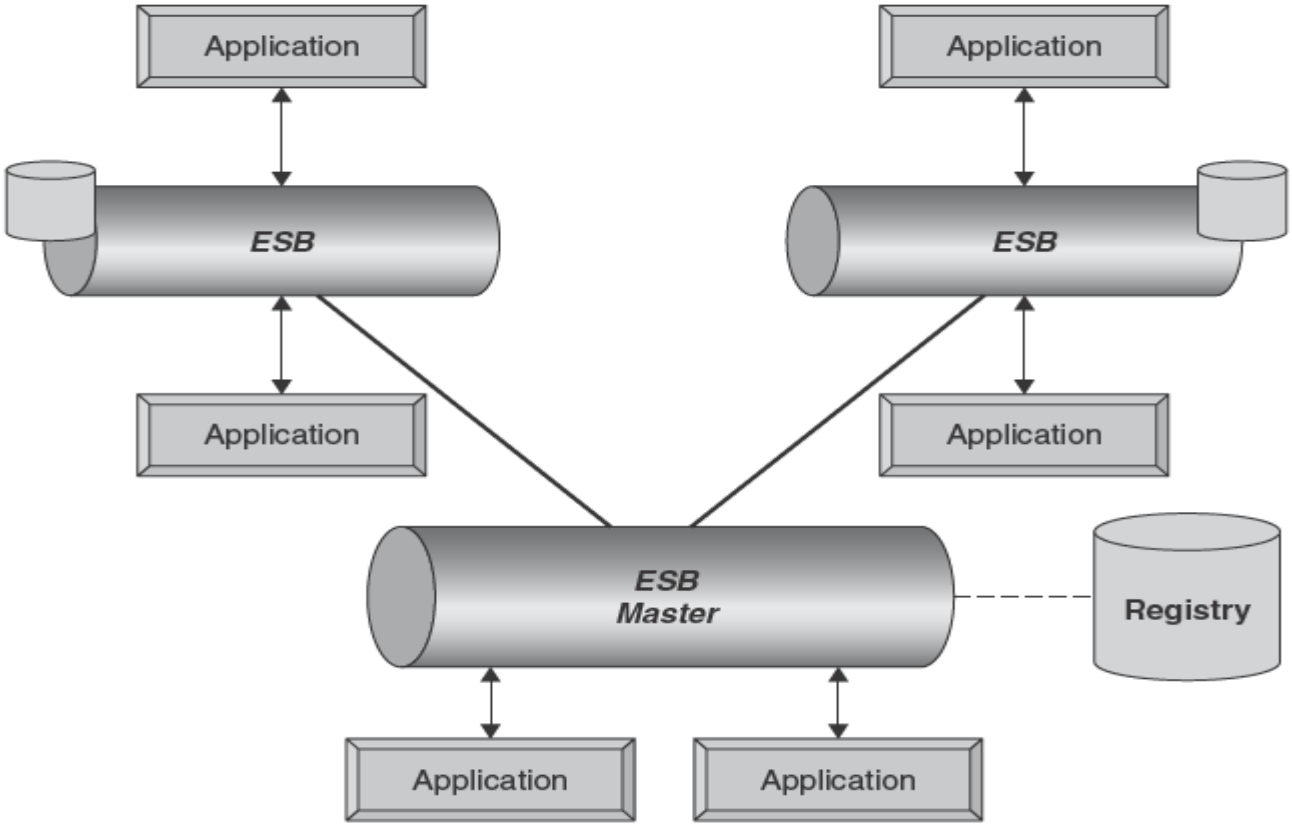


Global ESB

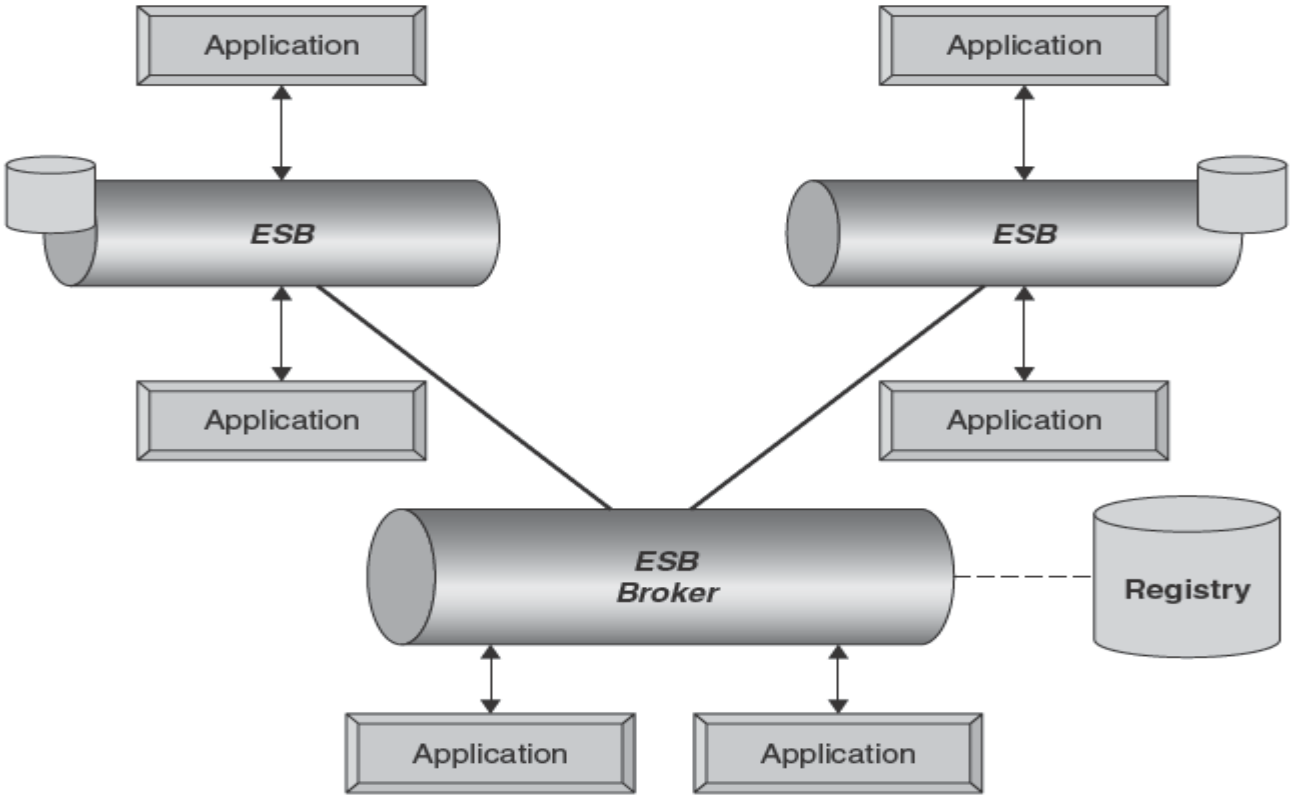
- Közvetlen összekapcsolású ESB
  - Több összekapcsolt ESB közös registry-t használ
  - Pl. ha SAP-t használ egy cég, akkor az SAP külön ESB-t használ (NewWeawer néven). Ilyenkor a meglévő ESB-t közvetlenül is lehet hozzákapcsolni



- Federated ESB
  - Az egyik ESB master/slave viszonyban van a többivel



- Brokered ESB
  - A bróker ESB közvetítő szerepet tölt be
  - Minden ESB-nek van saját registry-je, az interakciókat maguk kezelik



- ESB típusok
- Alkalmazás szerver alapú ESB
  - Az ESB az alkalmazás szerver egyik komponense (alkalmazása, pl JBOSS ESB)
  - Olcsó megoldás, könnyű telepíthetőség
  - Kevés alkalmazás integrációjára képesek
  - Alaptulajdonságai:
    - A programozó interakciókat hoz létre és telepíti őket
    - Sok operációs rendszeren futnak, java alapúak
    - Sok független szoftverbeszállító fejleszt adaptereket hozzájuk
    - Könnyen bővíthető federated modellre
- Üzenetkezelő rendszer alapú ESB
  - IBM WebSphere Message Broker - termékpélda
  - Rugalmasabban skálázható rendszerek, nagymennyiségű tranzakció lekezelésére képesek
  - Java, C++, COBOL alkalmazások könnyű integrációja is lehetséges
  - Telepítése bonyolultabb
  - Üzenet validáció, üzenet transzformáció bármilyen üzenetformátum között
  - Szolgáltatások virtualizációja
- Hardver alapú ESB
  - IBM WebSphere DataPower Appliances – termékpélda
  - Ilyenkor az ESB szoftvert egy speciális hardverrel együtt árulják
  - Natív registry összekapcsolás, közvetlen adatbázis összeköttetés

Felhő technológia

- A számítási felhő fogalom alatt azt értjük, amikor olyan alkalmazásokkal dogozik a felhasználó, melyek fizikailag egy távoli szerveren helyezkednek el, távoli erőforrásokat alkalmaz.
- A 60-as években már ismert fogalom volt, ekkor vetették fel azt az elképzelését, hogy a számítástechnikának közüzemi szinten kellene szerepelnie.
  - A felhasználó egy kis teljesítményű és tárhelyű eszközön egy böngészővel felcsatlakozik a szolgáltató szervereire, ahol igénybe veszi annak erőforrásait.
    - A szolgáltatóknál hatalmas kapacitású és mennyiségű hardver eszközökre van szükség ahhoz, hogy gyorsan és hatékonyan kiszolgálhassák az ügyfeleket, valamint azok adatait is tárolni tudják. A felhő erőforrás-megosztást is megvalósít és gyorsan alkalmazkodik a változó igénybevételhez
    - Hátránya lehet, hogy valójában nem tudjuk pontosan megmondani, hogy fizikailag a világ melyik részén tárolják az adatainkat és nem tudjuk biztosan azt sem, hogy illetéktelenek nem férnek-e hozzá rajtunk kívül.

Felhő technológia

Alapvetően három szolgáltatási modellt különböztetünk meg [3]:

- Szolgáltatásként kínált szoftver (Software-as-a-Service - SaaS): Eltérve a hagyományos megközelítéstől, a megrendelő nem egy telepíthető, birtokolható szoftvert kap, hanem az alkalmazásokat interneten keresztül használja. Ez által a cégek rugalmasan skálázhatják az alkalmazásaikat, nem kell nagy hardver-beruházásokat finanszírozniuk, üzemeltetésben, biztonságban jártas szakembereket alkalmazniuk.

- Szolgáltatásként kínált platform (Platform-as-a-Service - PaaS): Olyan platformot biztosít a fejlesztők számára, ahol gyorsan és rugalmasan együtt tudnak fejleszteni, valamint tárolási és számítási kapacitást biztosít a web-alkalmazásokhoz. Ez a modell az előző egyfajta kiegészítéseként is tekinthető. Sikeres SaaS szolgáltatásra példaként a salesforce.com-ot említhetjük.

- Szolgáltatásként kínált infrastruktúra (Infrastructure-as-a-Service - IaaS): Nemcsak alkalmazásokat és platformokat vehetünk igénybe szolgáltatásként, hanem erőforrásokat is. Például egy szolgáltatás operációs rendszert, tárterületet és távoli hozzáférést biztosíthat.

Felhő techológia

- Megkülönböztetünk privát és a nyilvános felhőket: a nyilvános felhőn több felhasználó is osztozhat, amíg a privát felhő valamilyen szervezet kizárólagos használatában van.
- Privát felhőt a saját belső (vagy kiszervezett) informatikai részleg üzemelteti. A nyilvános felhők a hatékonyság érdekében általában csak nagyon korlátozott szolgáltatást (Saas, PaaS), míg a privát felhők akár nagyszámú alkalmazást is és teljes platformokat is kínálnak (IaaS).
- A felhő technológia megjelenésével új eszközök és módszerek jelentek meg az informatikában, bizonyos szakterületeken.
  - Felhő architekt; Felhő adminisztrátor; Felhő szolgáltatás menedzser; Felhő adat architekt; Felhő tárhely adminisztrátor; Felhő alkalmazás architekt; Felhő operátor; Felhő felhasználó; Felhő fejlesztő.

Ubuntu Cloud szerver (UEC)

- Az Ubuntu Linux célul tűzte ki, hogy kiválasztja és folyamatosan karbantartja azokat a nyílt forráskódú eszközöket, amelyek segítségével privát felhők alakíthatók ki.
- Az Amazon EC2/3 API-t megvalósító Eucalyptus (http://open.eucalyptus.com/) projektre támaszkodnak. Az Ubuntu 9.04 szerverbe építették bele az Eucalyptus egy továbbfejlesztett változatát a KVM hipervisorral kiegészítve, és ezt a változatot UEC-nek (Ubuntu Enterprise Cloud) nevezték. Amikor a 10.10-es változat került a piacra, az UEC már a nyílt forráskódú privát felhők legsikeresebb változatává vált, amely már a hibrid felhők irányába is fontos lépéseket tett.

UEC komponensei

- **Node Controller (NC)**: Vezérli a virtuális gépek életciklusát. Ez a szoftverkomponens alap esetben, a KVM hypervisort, mint virtuális gépet alkalmazza. Az NC egyik oldalról a futtató operációs rendszerrel és a hypervisorral, másikról pedig a Cluster Controllerrel (CC) van kapcsolatban, továbbá a rugalmas bővíthetőség elősegítése miatt lekérdezi az CPU-k és a magok számát, a fizikai memóriát, a rendelkezésre álló lemezterületet, az aktuálisan futó virtuális géppéldányok számát és továbbítja a CC számára. Tehát összefoglalva, fő funkciói: az erőforrások karbantartása és a VM példányok életciklusának kezelése.

- **Cluster Controller (CC):** A CC menedzseli a Node Controllereket, telepíti a VM példányokat, és a példányok hálózatkezeléséért is felelős. Az Eucaliptus bizonyos típusú hálózati módokat alkalmaz például: IP kontroll, biztonsági csoportok, metadata szolgáltatás, VM izoláció, amelyeket a privát felhő topológiájának tervezésénél figyelembe kell venni. Összefoglalva a CC felelős azért, hogy egy új VM példány melyik NC segítségével induljon el, vezérli a hozzá tartozó virtuális hálózatot, továbbá összegyűjti az NC-k futási adatait és továbbítja a Cloud Controller (CLC) felé.

- **Walrus (W3):** W3 perzisztens egyszerű tároló szolgáltatást kínál REST és SOAP API-n keresztül. Tárolja a VM példányok fizikai állományait, ezek mentéseit (snap-shot), valamit egyszerű fájlkiszolgálóként használható a felhőben.

- **Storage Controller (SC):** Az SC támogatja a perzisztens block szintű tárolók alkalmazását. Az Amazon Elastic Block Store (EBS) megvalósításának tekinthető. Ez a tároló független az őt használó VM példányoktól, azok használják, amelyeknek szüksége van adatbázisra, fájlrendszerre vagy blokk szintű tárolóra. A tárolók mérete dinamikusan növelhető, ami kellő rugalmasságot ad a használata során.
- **Cloud Controller:** A privát felhő infrastruktúra front-end komponense a Cloud Controller (CLC), amely egy webszolgáltatás interfész. Egyik oldalról a kliensek az API-n keresztül elérhetik szolgáltatásit, másik oldalról a fenti komponensekkel van kapcsolatban. Web-es interfészt is tartalmaz, és monitorozza a felhő erőforrásait VM példány szinten, továbbá eldönti, hogy egy új példány melyik CC segítségével induljon el.

Mintafelhő konfiguráció