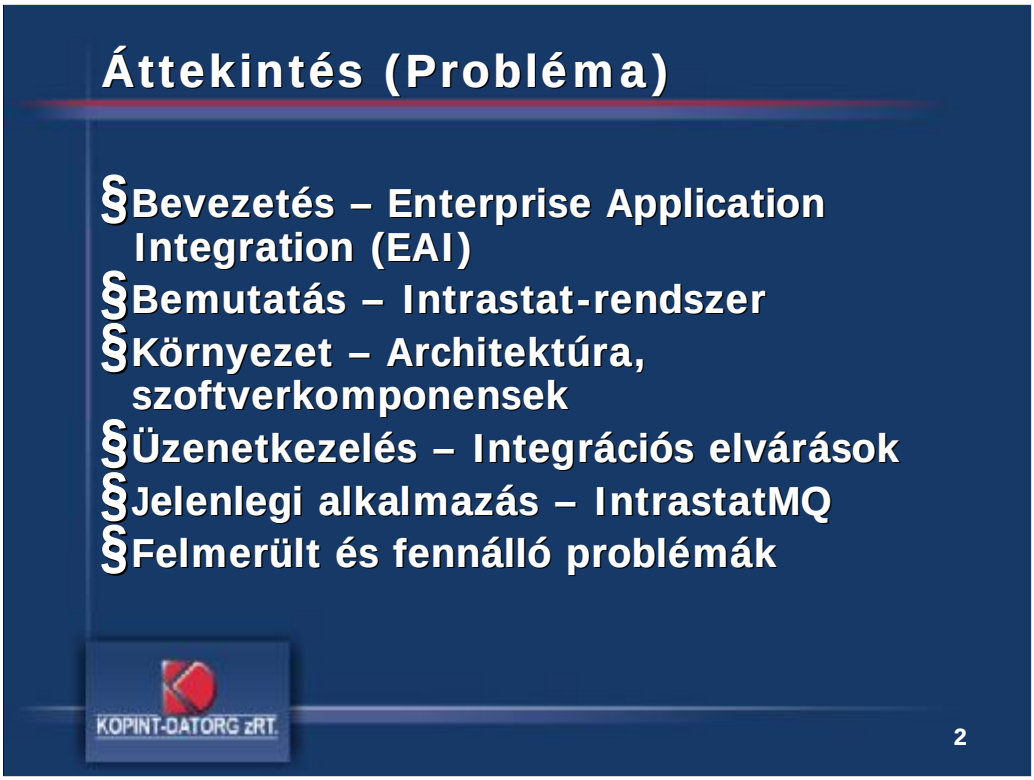

# És felszáll a köd: alkalmazásintegrációs megoldás fejlesztése a gyakorlatban Oracle Application Server és IBM WebSphere MQ környezetben

Viczián István
Vezető szoftverfejlesztő
viczian@kopdat.hu
Kopint-Datorg Zrt.
http://delfin.klte.hu/~vicziani

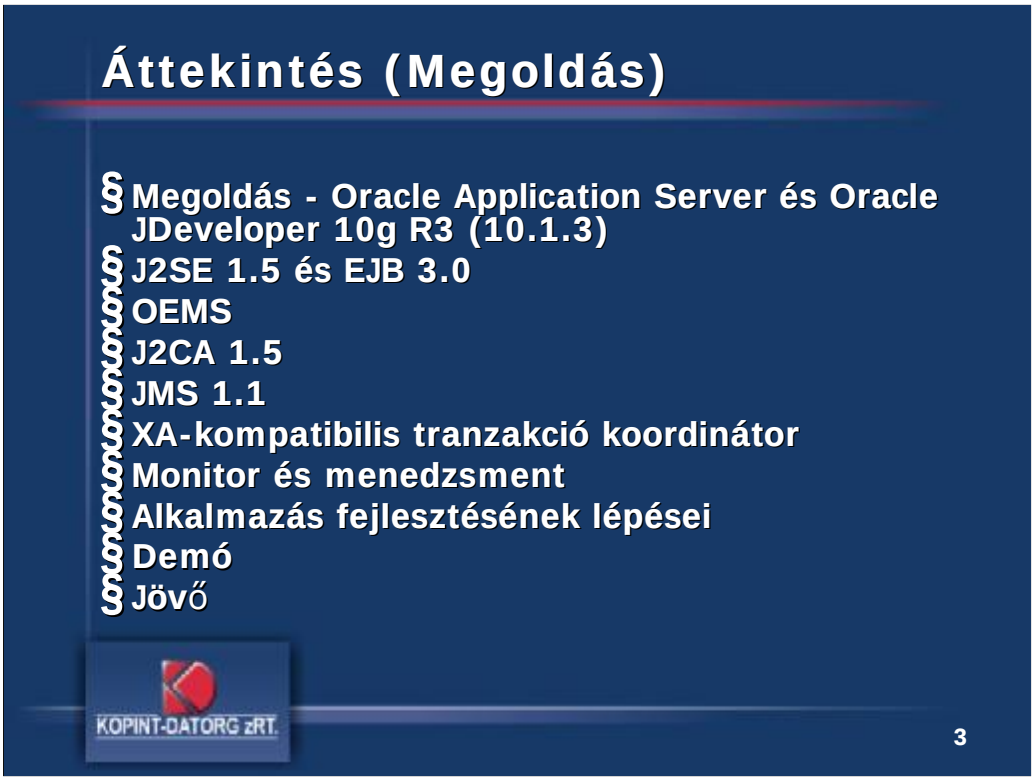



Napjainkban elterjedt informatikai fogalom a nagyvállalati alkalmazásintegráció (Enterprise Application Integration - EAI), valamint a szolgáltatásorientált architektúra (Service Oriented Architecture – SOA), pontos definíció mégsem létezik egyikre sem, mindegyik gyártó egy kicsit másképp, sajátosan értelmezi. Különösen akkor kerül ez előtérbe, mikor egyszerre több gyártó termékét kívánjuk használni. A fogalmak gyakori és talán nem mindig megfelelő használata, valamint a területen tapasztalható sokszínűség, változékonyság, dinamizmus lehet az alapja annak, hogy aki nem kíséri nyomon a különböző ajánlásokat, szabványokat és termékeket, könnyen elveszhet az ezek által keltett ködben. Előadásom célja ezeket a fogalmakat tisztázni egy konkrét megoldás bemutatásával.

Előadásom alapját a Központi Statisztikai Hivatal belső rendszereit és a KSH megbízásából a Kopint-Datorg Rt. által 2004 óta fejlesztett és üzemeltetett Intrastat-rendszert integráló megoldás adja.

Célom egy konkrét példán keresztül bemutatni, hogy egy Oracle Application Server és IBM WebSphere MQ alapú környezetben hogyan lehet egy alkalmazásintegrációs megoldást megvalósítani, tisztázva az alapfogalmakat, lehetőségeket.

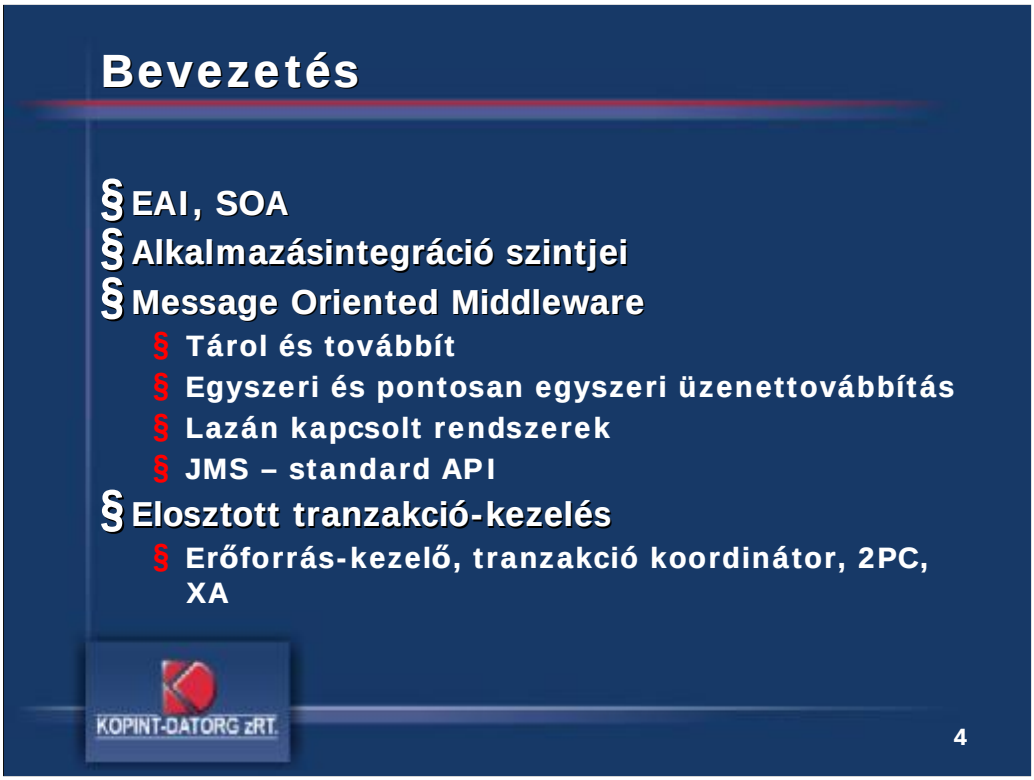
Előadásom során bemutatom a rendszer környezetét, mind az architektúrális felépítését, mind a rendszert felépítő szoftver komponenseket. Ismertetem a rendszer fejlesztésének kezdetén felmerült igényeket és elvárásokat, valamint az implementáció eredményeként előállt rendszert, és az alkalmazott technológiákat. Az implementáció és üzemeltetés közben felmerült nehézségek és problémákat is felsorolom, melyek közül többet sikerült megoldani, de vannak olyanok is, melyek még mindig megoldásra várnak.




A 2006. január 26-án bejelentett OC4J 10g (10.1.3) pehelysúlyú konténer és rá épülő alkalmazásszerver, valamint a JDeveloper 10g (10.1.3) fejlesztőeszköz megjelenése adott alkalmat arra, hogy felülvizsgáljuk a jelenlegi alkalmazást, és összevessük az újonnan megjelent eszközök által támogatott friss technológiákkal. Itt kiemelném az új nyelvi elemekkel is bővült J2SE 5.0 környezetet, az egyszerűbben kezelhető EJB 1.3-as szabvány komponenseit, az egységes integrációs platformot nyújtó OEMS környezetet, az a J2EE Connector Architecture szabványt - melynek révén már meglévő, beépített konnektorokkal kapcsolódhatnak az Oracle (akár OracleAS JMS, akár OJMS) és 3rd Party JMS providerek -, az XA-kompatibilis tranzakció koordinátort, a JMS 1.1 szabványt, valamint a monitorozást és menedzsmentet megkönnyítő új funkciókat és szabványokat.

A teljes megértés kedvéért kifejtem az alkalmazás fejlesztésének lépéseit, sőt betekintést nyújtok a kódokba és konfigurációs állományokba (telepítés leírók), valamint a rendszert élő demó keretében működés közben is megmutatni, így testközelivé hozva az alkalmazásintegrációt, és megmagyarázni a hangzatos, de titokzatos szavakat, betűkombinációkat. Az előadás így főleg fejlesztőknek, vezető fejlesztőknek és a technológia iránt mélyebben érdeklődőknek szól, akik kíváncsiak, mi is történik a színfalak mögött.

Előadásom végén bemutatom a lehetséges továbbfejlődési irányvonalakat, főbb technológiákat.

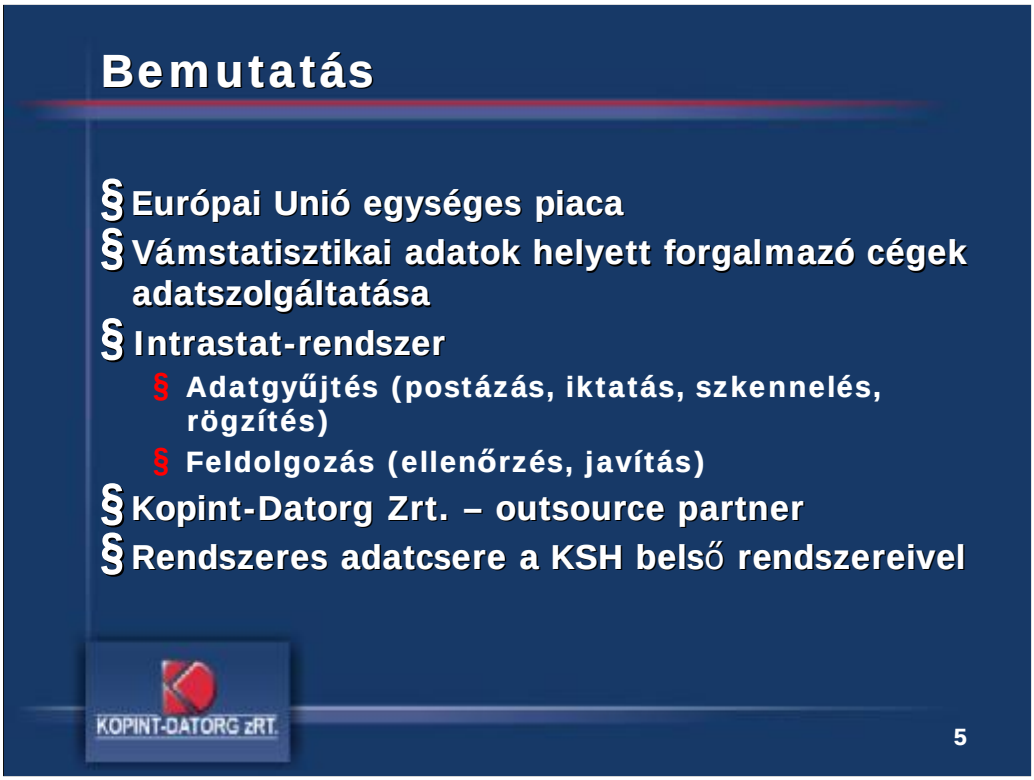



Az alkalmazásintegráció több szintre tagolódik, ennek legalsó szintje az adat szintű integráció, ahol az alkalmazások között kell adatot átvinni. Ennél magasabb szintű integráció az alkalmazás szintű integráció, ahol az alkalmazások a normál működés közben egymással kommunikálnak. Következő szint a folyamat szintű integráció.

Előadásomban egy, az első két szint között meghúzódó integrációs alkalmazást fogok bemutatni. Az integráció a nagy múlttal rendelkező üzenetközpontú köztesalkalmazás (message oriented middleware) réteget használja fel integrációs megoldásként. A köztesalkalmazás egy manapság gyakran használatos divatos fogalom, definíciója mégsem olyan egyértelmű. A köztesalkalmazás egy olyan elérhető szoftverréteg, mely a heterogén platformok és protokollok hálózati rétege és az üzleti alkalmazások között helyezkedik el. A köztesalkalmazások egyik csoportja az üzenetközpontú köztesalkalmazások. Ezen köztesalkalmazások alapfogalmai az üzenet (fej, törzs) és a sor. Segítségükkel szinkron és aszinkron kommunikáció is megvalósítható. A tárol és továbbít modellnek megfelelően a sorba az egyik alkalmazás beleteszi az üzenetet, és a másik alkalmazás akkor veszi ki, amikor erre ideje lesz.

Így lazán kapcsolt rendszerek implementálhatóak. Az üzenet sorakoztató köztesalkalmazás feladata az egyszeri és pontosan egyszeri üzenettovábbítás. A Java világban a szabványos API a JMS. A JMS csak egy programozói interfész, de a mögötte rejlő funkcionalitást már egy message provider végzi. Perzisztens sorok valók az üzenetek megbízható tárolására. Egy message provider részt vehet elosztott tranzakciókban is.

Elosztott tranzakciókról akkor beszélünk, ha egy tranzakcióban több erőforrás-kezelő (resource manager) is részt vesz (pl. több adatbázis, vagy üzenet sorakoztató köztesalkalmazás). A tranzakció-kezelés az adatok integritását hivatott megőrizni. Az elosztott tranzakciót egy központi elem végzi, melynek neve tranzakció koordinátor. Az elosztott tranzakció-kezelés alapja a 2PC protokoll. A tranzakció koordinátor és a resource provider-ek közötti kommunikáció szabványa az XA protokoll.




2004. május 1-én Magyarország csatlakozott az Európai Unió egységes piacához. A vámhatárok megszűnése következtében a termékek mindenfajta adminisztratív megkötöttségek nélkül szabadon mozoghatnak. Statisztika készítéséhez az eddigi vámstatisztikai adatok helyett közvetlenül a

forgalmazó cégektől kell adatot gyűjteni. Ezen adatgyűjtést EU jogszabályok (rendeletek) szabályozzák, betartása minden tagállam számára kötelező.

Ezen adatgyűjtést az Intrastat-rendszer valósítja meg, melynek működtetését a Központi Statisztikai Hivatal végzi.

A rendszer feladata az ügyfél adatok kezelése, az adatok gyűjtése (postázás, iktatás, szkennelés, rögzítés) és feldolgozása (ellenőrzés, javítás).

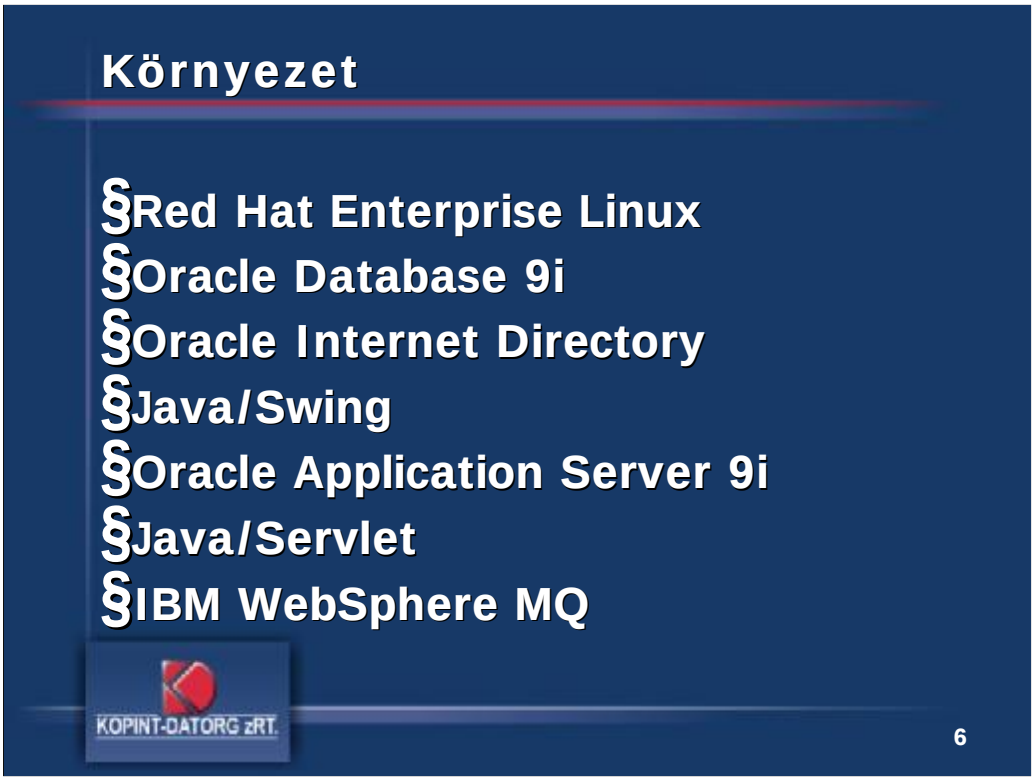
A KSH feladata a kész statisztikák hazai publikálása, mely a piaci szereplők üzleti döntéseihez lehetnek szükségesek, valamint az adatok továbbítása az EU statisztikai hivatalának.

Az Intrastat adatszolgáltatói azok a magyarországi vállalatok/vállalkozások, amelyek a EU-tagállam(ok)ból terméket hoznak be, vagy oda terméket visznek ki, valamint egy évre vonatkozó EU beérkezése vagy kiszállítása meghaladja az un. adatszolgáltatási küszöbértéket.

Az adatok beadása lehet papír alapú, valamint elektronikus (e-mail-ben vagy az e-star rendszerben továbbított).

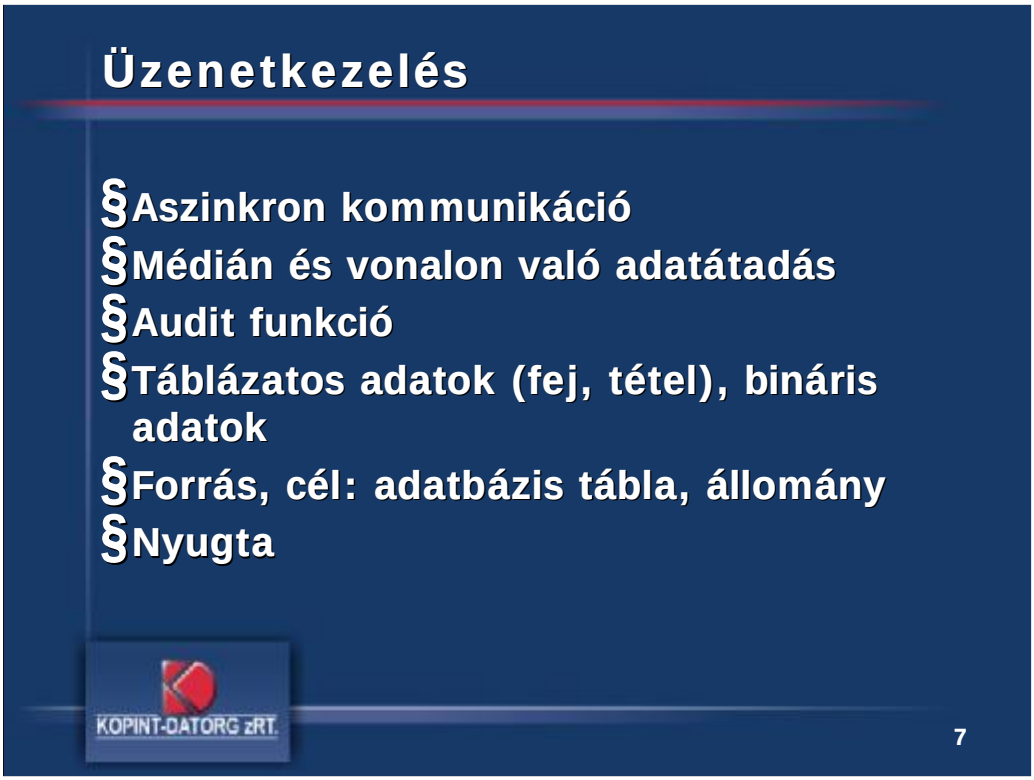
Az Intrastat-rendszert a KSH megbízásából a Kopint-Datorg Zrt. fejleszti és üzemelteti.

Az Intrastat-rendszer fizikailag a Kopint-Datorg Zrt. székhelyén helyezkedik el, és rendszeresen adatot kell cserélnie a KSH belső rendszereivel. Ezen kívül a KSH számára webes felületen keresztül monitoring funkciókat kell biztosítani.

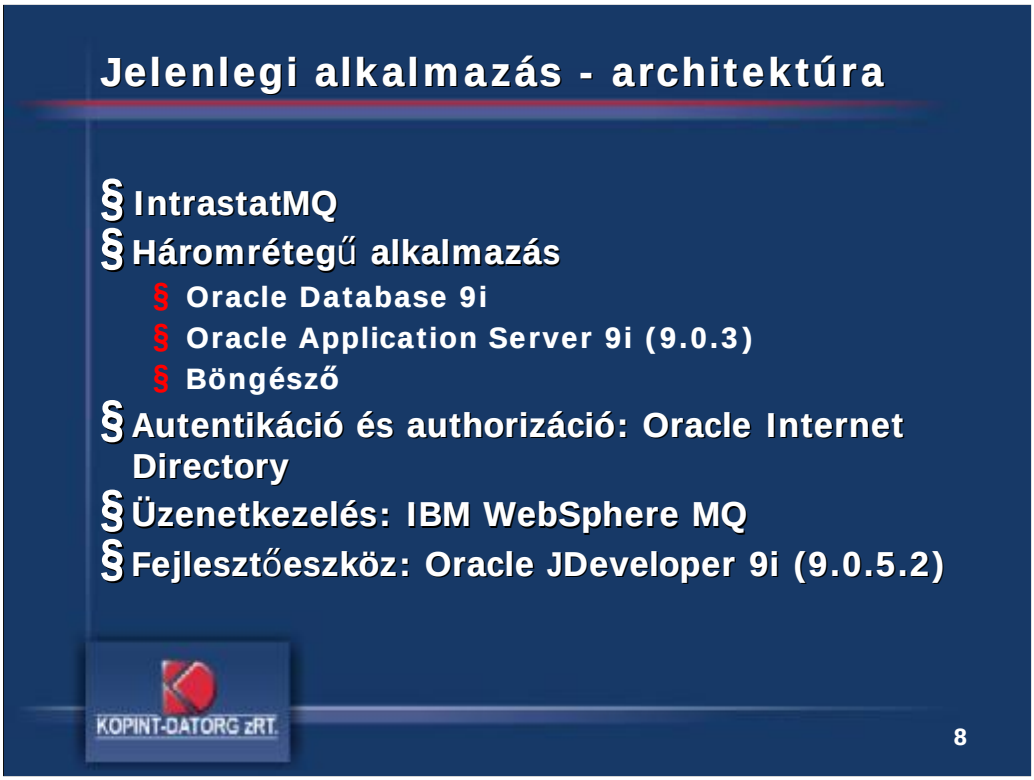



A Kopint-Datorg Zrt. oldalán széleskörűen alkalmazott operációs rendszer a Red Hat Enterprise Linux. Az Intrastat-rendszer klasszikus kliens-szerver alkalmazások együttese, ahol a szerver egy Oracle Database 9i, mely az adatok perzisztenciájáért felelős, és a kliens alkalmazások Java nyelven fejlesztett vastag kliensek Swing technológiával. A felhasználók nyilvántartása Oracle Internet Directory (LDAP implentáció) címtár segítségével valósul meg.

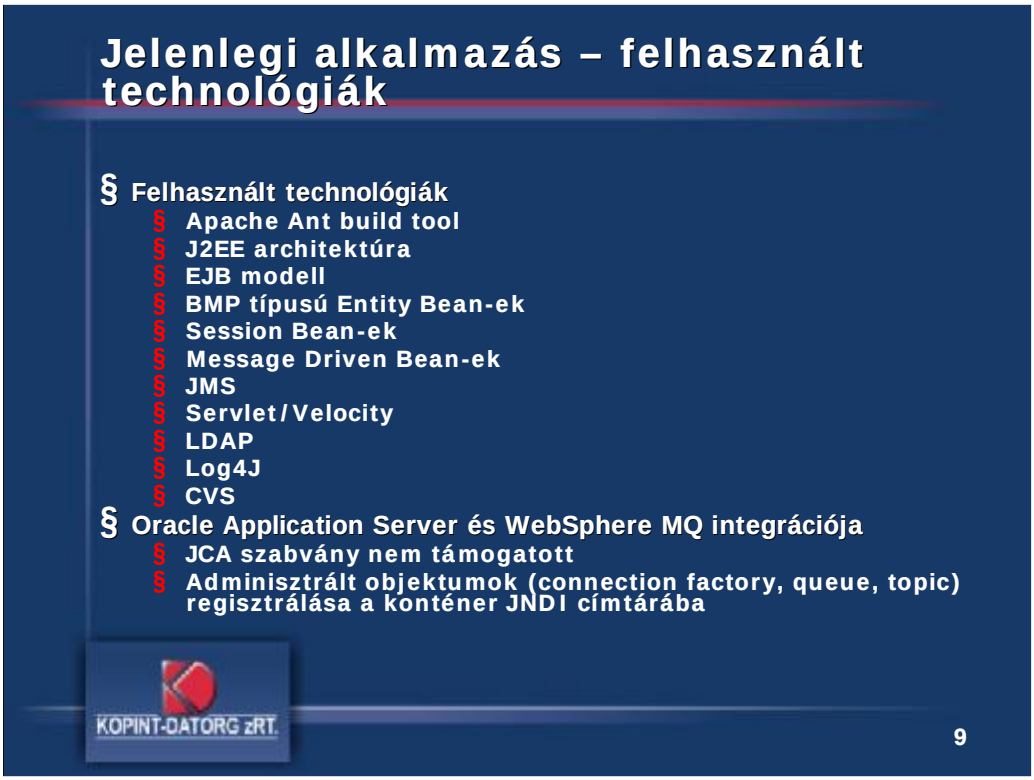
A Kopint-Datorg Zrt. és a KSH között egy menedzselt bérelt vonal került kialakításra az adatkapcsolatot megvalósítandó. A kommunikációs middleware a KSH-ban már elfogadott és széleskörűen alkalmazott IBM WebSphere MQ (korábban MQSeries). A kommunikáció biztonságát az IBM WebSphere MQ SSL titkosítása szavatolja, mely miatt annak 5.3-as verziójára kellett migrálni a KSH oldalán, hiszen ez a funkció csak ezen verzióban került bevezetésre. Az alatta elhelyezkedő protokoll szabványos TCP/IP protokoll.




A kommunikáció alapvetően aszinkron. A gyakran átadandó, nem nagy mennyiségű adatok továbbítása vonali kapcsolaton, a ritkán átadandó, nagy mennyiségű adatok átadása médián keresztül történik. A kettő között egy adattípusnál az átállás megoldható. Az érkezett adatok feldolgozásakor nem szignifikáns, hogy az azok milyen forrásból érkeztek, viszont részletes nyilvántartást kell vezetni, hogy milyen adatok mikor kerültek átadásra (audit funkció). Az adatok alapvetően táblázatos (tétel) adatok, kiegészítve különböző fej információkkal (meta-adatok). Ezeket szintaktikailag ellenőrizni kell (oszlopok száma, oszlopok típusa). Ezen kívül lehetőség van tetszőleges bináris állományok (dokumentumok, képek) azonnali átvitelére is. Az átvitt adat forrása és célja lehet adatbázis tábla, valamint állomány is. Az átvitt adatok megérkezésekor és szintaktikai ellenőrzésének eredményéről nyugtát kell visszaadni. Médián átvitt adatok esetén is vonalon kell ezt a nyugtát visszaadni.




Az adatcseréért felelős alkalmazás egy logikailag külön egység, mely az Intrastat-rendszer többi moduljával adatbázis táblákon keresztül kommunikál. Az alkalmazás egy Java-ban fejlesztett, klasszikus három rétegű alkalmazás, ahol a perzisztenciát az Oracle Database 9i biztosítja, az alkalmazás logikának környezetet az Oracle Application Server 9i (9.0.3) ad, és a prezentációs réteg az internet böngésző, mely a felhasználóval tartja a kapcsolatot. A felhasználói azonosításhoz és jogosultságkezeléshez az felhasználói adatokat az Oracle Internet Directory adja. Az üzenetközpontú köztesréteg az IBM WebSphere MQ, melynek egy sorkezelője (queue manager) a Kopint-Datorg Zrt.-nél fut, egy másik a KSH-nál. A fejlesztői, teszt és éles környezet szeparálása külön sorkezelők telepítésével történik. A fejlesztés Oracle JDeveloper 9i (9.0.5.2) integrált fejlesztő környezettel történt. Különböző típusú üzenetek, valamint a nyugták számára külön perzisztens sorok lettek definiálva, irányonként egy. Az egyszeri és pontosan egyszeri üzenetküldés biztosítására mind a lokális sorkezelőn definiálva lett egy sor (lokális sor), mind a távoli sorkezelőn egy neki megfelelő sor (távoli sor). A lokális és távoli sorok un. csatornákkal vannak összekötve.




Az alkalmazás fejlesztése során felhasznált technológiák:

•Apache Ant build tool: alkalmazás fordítására, összeállítására, telepítésére.

•J2EE architektúra: szabványos komponens alapú architektúra, ahol az alkalmazásszerver biztosít környezetet a nagyvállalati komponensek számára.

•EJB modell: szabványos nagyvállalati komponensek tervezésére és váz-generálására használatos eszköz.

•BMP típusú Entity Bean-ek: Bean menedzselt perzisztenciát megvalósító üzleti objektumok.

•Session Bean-ek: üzleti folyamatokat biztosító komponensek.

•Message Driven Bean-ek: aszinkron értesítéseket feldolgozó komponensek.

•JMS: Java alapú szabványos API üzenetek kezelésére.

•Servlet/Velocity: webes megjelenési réteget biztosító technológiát, felhasználva a sablon kezelést.

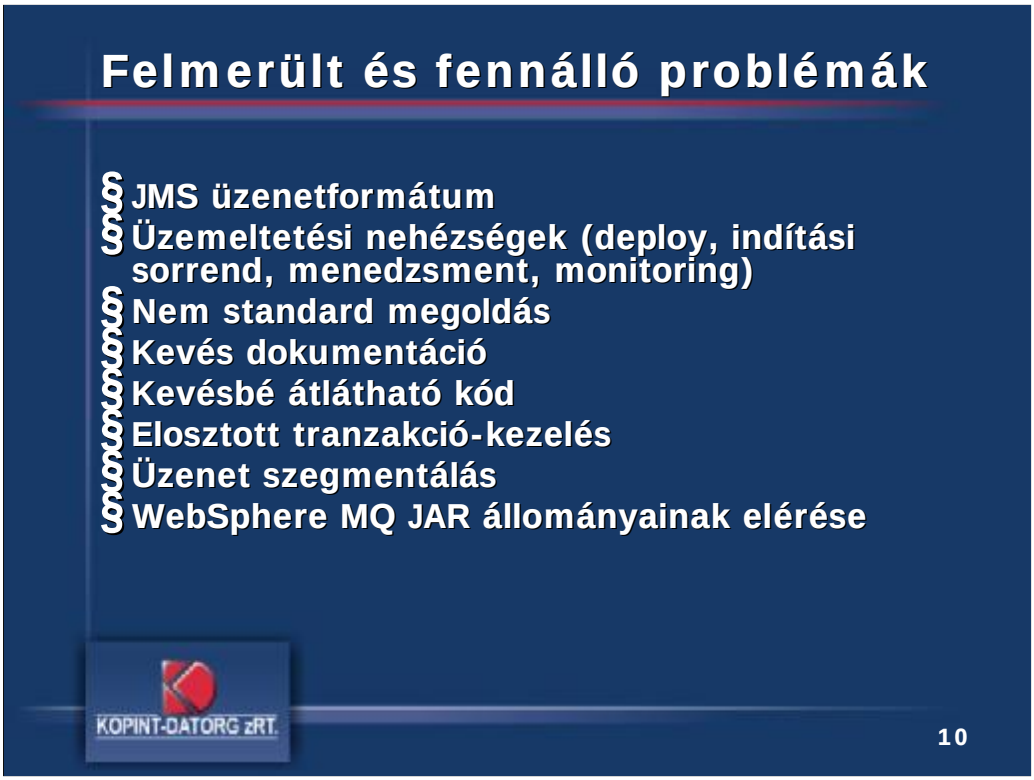
•LDAP: címtár kezelése felhasználói bejelentkeztetéshez és jogosultságkezeléshez.

•Log4J: kvázi standard naplózás.

•CVS: verziókezelés.

KSH oldalán használt küldő/fogadó alkalmazás C++ nyelven van implementálva.

A jelenleg használt alkalmazásszerver még nem támogatja a JCA szabványt, így az alkalmazásszerver és a message provider összekapcsolása viszonylag körülményes és nem szabványos művelet. A jelenlegi alkalmazásban ez úgy került megvalósításra, hogy először elindítjuk a konténert, majd annak JNDI címtárába regisztráljuk az WebSphere MQ objektumokat (connection factory, queue, topic), majd elindítjuk az alkalmazást, ami a JNDI nevek alapján használatba veszi az erőforrásokat.




Ezen megoldás fejlesztésekor több problémával is találkoztunk, melyek egy része megoldódott, más része viszont még megoldásra vár. Ezen problémák a következők:

•JMS üzenetformátum: a túloldalt levő C++ nyelvű alkalmazás nem tudta értelmezni a JMS üzeneteket. A WebSphere MQ-ban beállítható, hogy a JMS plusz információkat távolítsa el.

•Üzemeltetési nehézségek: körülményes deploy folyamat, ahol figyelni kell a műveletek sorrendjére, nehezen menedzselhető és monitorozható. A fejlesztő és a telepítő szerepköre egybemosódik.

•Nem standard megoldás, így szükség esetén kevésbé cserélhető.

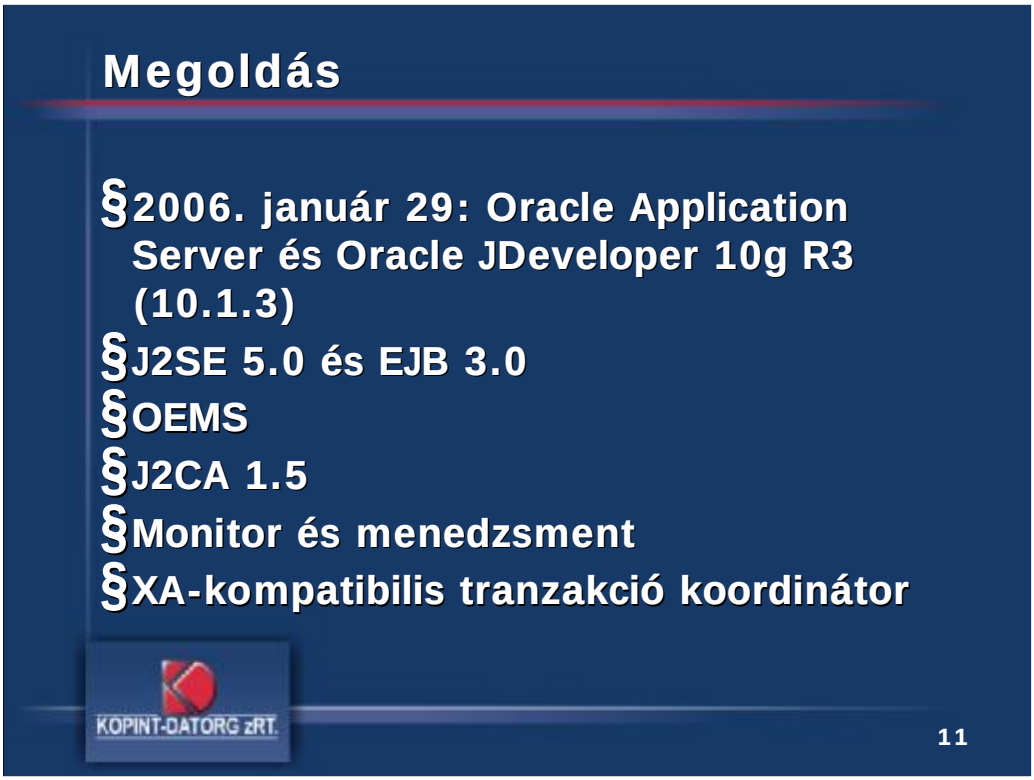
•Minimálisan dokumentált. Ez a megoldás ebben a formában nem is szerepelt sehol.

•Kevésbé átlátható kód (rengeteg Bean osztály és interfész, valamint telepítés leírók).

•Elosztott tranzakció-kezelés hiánya (nem sikerült konfigurálni).

•A JMS API nem támogatja az MQ üzenet szegmentálás szolgáltatását.

•Az MQ JAR állományait elérhetővé kell tenni az OC4J konténeren belül.




2006. január 29-én jelentették be az Oracle Application Server és Oracle JDeveloper 10g R3 (10.1.3) szoftvereket. Béta verziójuk már hosszabb ideje elérhető volt, így az új funkcionalitásokat már régebben ki lehetett próbálni. Az új verziók olyan mértékű újításokat tartalmaznak, melyek alapján érdemes újra vizsgálni jelenlegi alkalmazást,

és vagy migrálni, vagy a következő hasonló alkalmazásnál a tapasztalatokat felhasználni.

A következő újdonságokat érdemes megvizsgálni a jelenlegi integrációs megoldás szempontjából:

•J2SE 5.0

•EJB 3.0

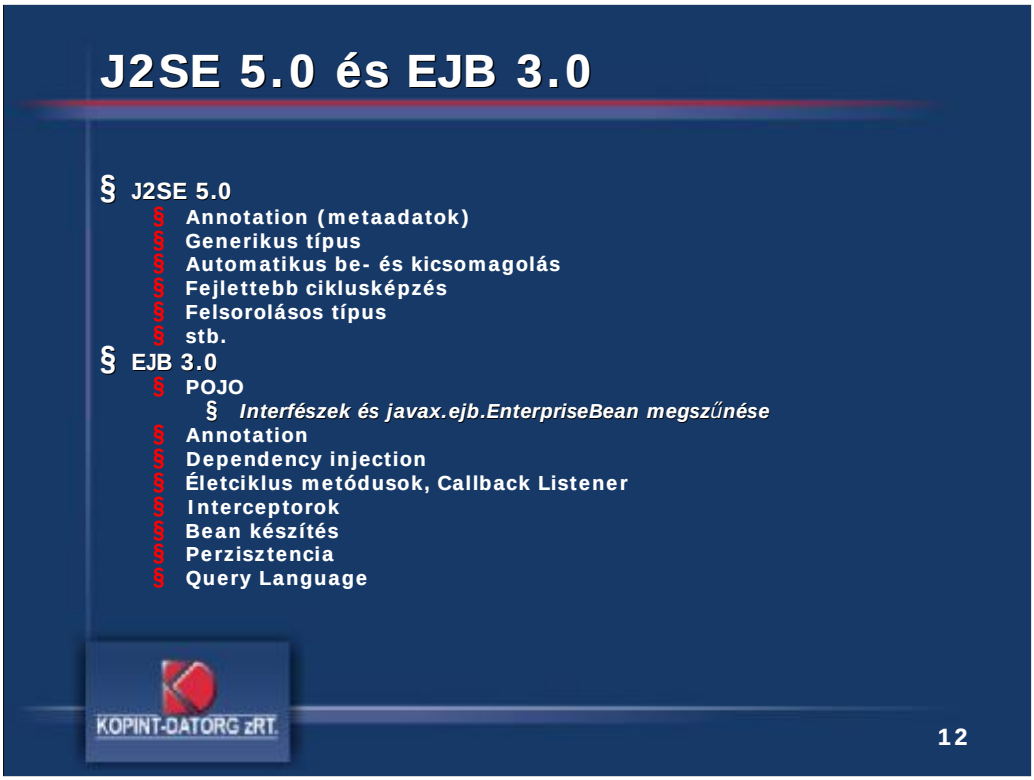
•Oracle Enterprise Messaging Service

•J2CA 1.5

•Monitor és menedzsment

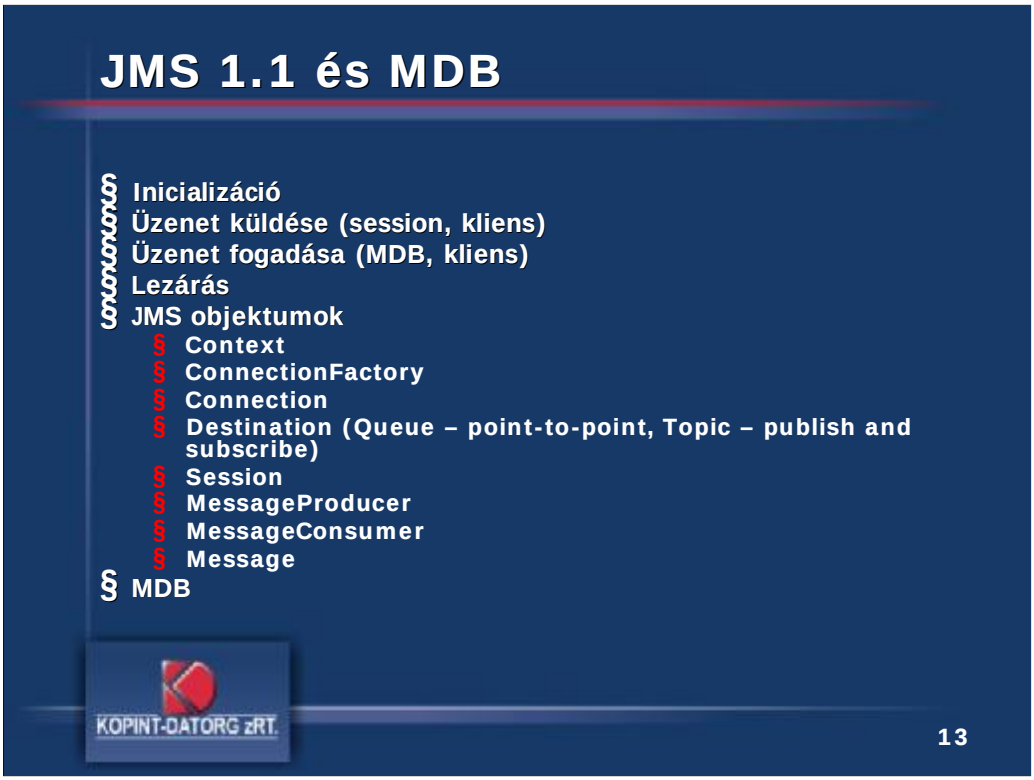
-XA-kompatibilis tranzakció koordinátor, tulajdonságai:

- •inter-positioning transaction inflow
- •last resource commit: nem XA kompatibilis resource provider is részt vehet elosztott tranzakcióban
- •transaction recovery log



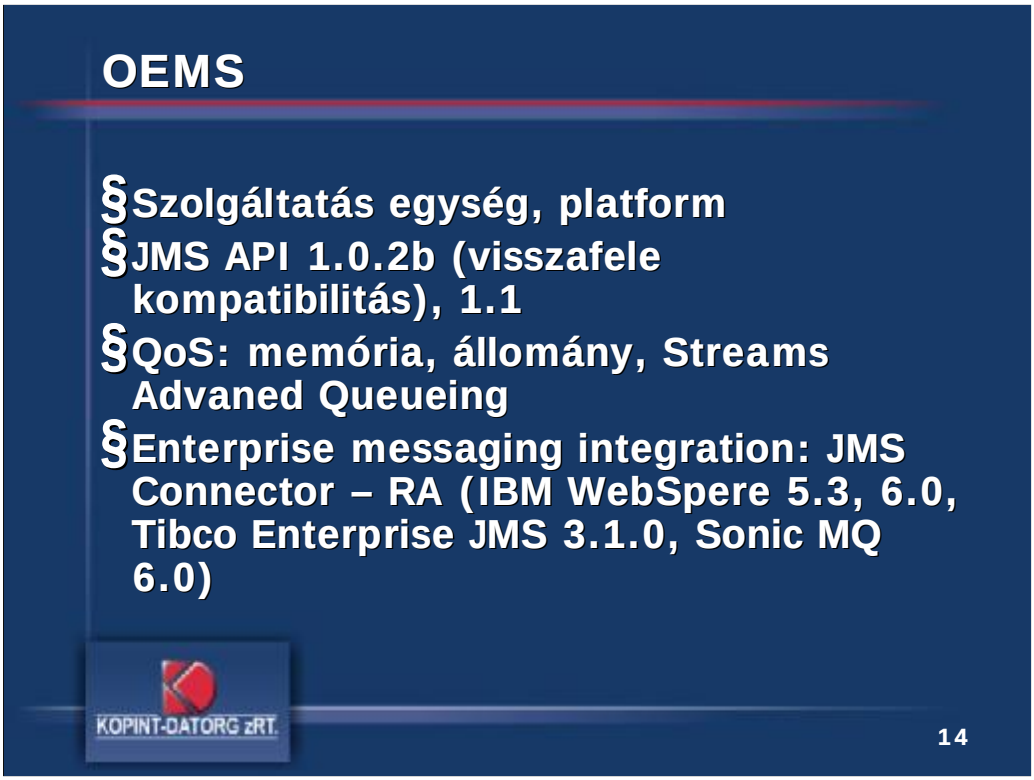

A J2SE 5.0 a Java nyelv és fejlesztési keretrendszer egyik legnagyobb módosítást tartalmazó verziója. Nem csak az osztálykönyvtárak változtak és bővültek, hanem új nyelvi elemek is bekerültek. A metaadatokat a Java forrás szövegében helyezhetjük el, és elérhetővé válik a fordító, illetve automatikus eszköz számára, sőt futás időben akár Reflection API-n keresztül is. Segítségével generálhatók különböző leíró és konfigurációs állományok, interfészek. A generikusokkal osztályok, interfészek és metódusok paraméterezhetők, egyel magasabb absztrakciós szintet képviselnek. Az automatikus be- és kicsomagolással a Java primitív típusú, valamint osztály típusú változói közötti értékadás egyszerűsödik. A fejlettebb ciklusképpzéssel a generikusok használatának ötvözésével egyszerűbb kódot írhatunk, megtartva a szigorú típusosságot. Ezen verzióba került be a felsorolásos típus is.

Az EJB 3.0 (JSR 220), mely a következő, 5.0-ás J2EE platform része, bevezetésének célja az EJB fejlesztés egyszerűsítése. Az EJB-khez ezentúl nem kell lokális és távoli interfészeket deklarálni, valamint nem kell a javax.ejb.EnterpriseBean interfészt implementálniuk. Annotation-ök használata a telepítés leírók helyett egy alternatíva, a forráskódban lehet különböző tulajdonságokat definiálni, pl. tranzakció és biztonsági beállítások, O-R megfeleltetések, valamint hivatkozásokat különböző környezeti vagy erőforrás referenciákra. A telepítés leíróban ezek a beállítások felülbírálhatóak. A dependency injection különböző környezeti vagy erőforrás referenciák használatának egyszerűsítését teszi lehetővé, azáltal, hogy nem nekünk kell lekérni, hanem annotation-nel deklarálva vannak. Az életciklus metódusok kezelése is egyszerűbbé válik, nem kell minden metódust implementálni, csak amelyikre szükségünk van (callback metódusok). Callback Listener-ekkel is meg lehet oldani az életciklushoz tartozó események figyelését. Az interceptorok olyan metódusok, melyek elkapnak üzleti metódus hívásokat. Ezen metódusok a bean-ekben és külön interceptor osztályokban is definiálhatóak. A bean készítés is egyszerűsödik, a JNDI-ből nyert referencián azonnal lehet metódushívásokat elvégezni, nem kell a create metódussal új bean példányt gyártani. Az Entity Bean-ek is sima POJO objektumok, melyek tetszőlegesen támogatják a polimorfizmust és öröklődést. EntityManager API használható adatbázisban tárolt egyedek lekérésére, létrehozására, törlésére és módosítására. Ez az API támogatja az objektumok lekapcsolását az adatbázisról, majd a módosítások visszaszinkronizálását. Az API használható mind konténeren belül, mind konténeren kívül. A Query Language is jelentős mennyiségű újítást tartalmaz, úgymint a nagymennyiségű (bulk) adatbázis műveleteket, JOIN, GROUP BY HAVING operációkat, projekciókat és allekérdezéseket (sub-query). Valamint dinamikusan is összeállíthatóak lekérdezések, nem csak a telepítés leíróban.

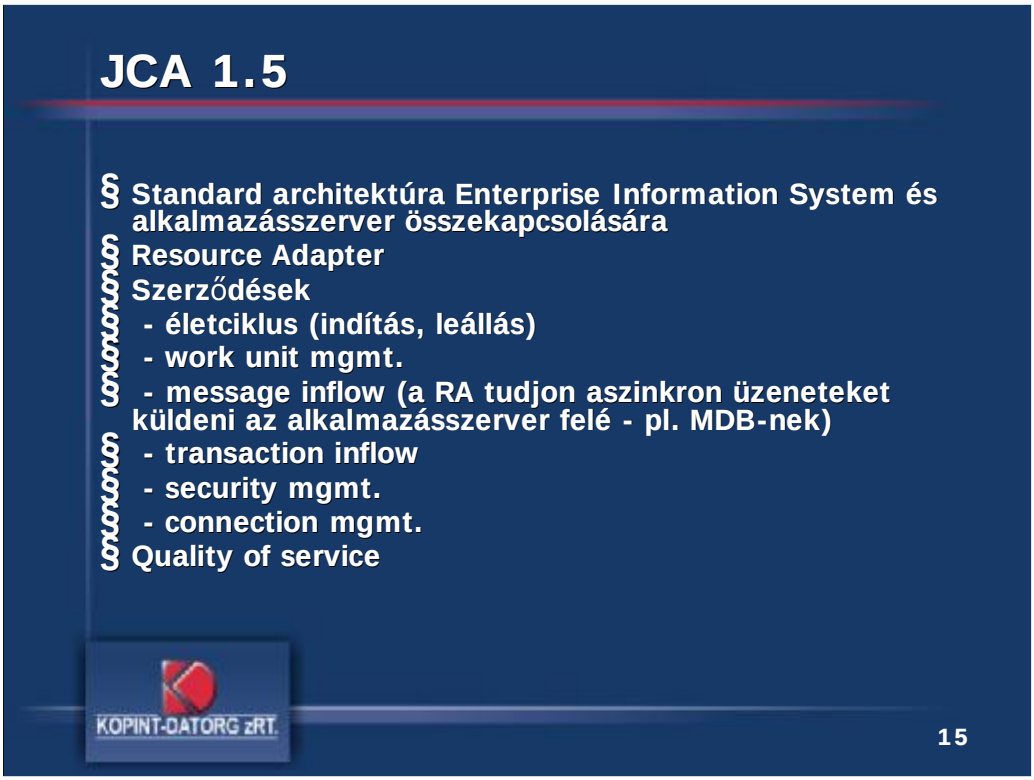



A JMS API egy szabványos gyártó független programozói interfész, mely elérhetővé teszi a message provider-t. Ahhoz, hogy üzenetet küldjünk vagy fogadjunk, bizonyos inicializációs lépéseket meg kell tennünk, fel kell építenünk a kapcsolatot az erőforrással. Bizonyos objektumokat az implementáció és a futtatási környezet teljes különválasztása miatt JNDI címtárból keresünk ki, ehhez szükség van a környezetre (Context). Logikai név alapján ettől kérjük el a ConnectionFactory, valamint a konkrét sor (Queue) vagy téma (Topic) objektumokat. Ponttól pontig (point-to-point) kommunikáció esetén az előzőt, közzétesz és előfizet modell esetén az utóbbit. Mindkettő a cél (Destination) leszármazottja. A ConnectionFactory objektummal gyárthatunk kapcsolat (Connection) objektumokat, majd ezzel munkamenet (Session) objektumokat. A Session és Destination objektumok segítségével készíthető MessageProducer üzenetek küldésére, MessageConsumer üzenetek fogadására. Üzenet küldést általában Session Bean-hez kapcsolódó segédosztályokkal, vagy klienssel végzünk, míg üzenet fogadását J2EE környezetben Message Driven Bean-nel, különálló alkalmazásban közvetlenül a klienssel. Az üzeneteknek több fajtája van, melyekkel küldhetünk szöveget, bájtsorozatot, név-érték párokat, stb.

A Message Driven Bean-ek esetén az inicializálást nem kell elvégezni, ezt a konténer megteszi helyettünk, emellett infrastruktúrát nyújt tranzakciók és biztonság kezelésére, valamint végzi a párhuzamos feldolgozást.




Oracle Enterprise Messaging Service az R3-ban került bevezetésre, mely egységesíti a message provider-ek kezelését. Nem egy konkrét szoftver, hanem az az adatbázis és az alkalmazásszerver üzenetkezelő és integrációs szolgáltatásainak egysége, melyen fejleszteni, és melyre telepíteni lehet alkalmazásokat SOA környezetben. Az OEMS a JMS API 1.0.2b (visszafele kompatibilitás miatt), és 1.1 verzióját biztosítja az alkalmazás fejlesztőinek üzenet alapú szolgáltatások elérésére. Így a fejlesztőnek nem kell foglalkoznia az alatta lévő infrastruktúrával, ami az alkalmazás módosítása nélkül módosítható a változó igényeknek megfelelően. A message provider a JMS API alatt lehet Oracle megoldás, mint az alkalmazásszerverben futó OracleAS JMS, valamint az adatbázisba épített Streams Advanced Queueing. Az OracleAS JMS képes az üzeneteket memóriában, perzisztens esetben fájlban is tárolni. A SAQ az üzeneteket adatbázisban tárolja. Az OJMS a SAQ-hoz használt JMS interfész neve, és nem keverendő az OracleAS JMS-sel. Az Oracle által szállított JMS Connectorral azonban 3rd party message provider-ek is csatlakoztathatóak (IBM WebSpere 5.3, 6.0, Tibco Enterprise JMS 3.1.0, Sonic MQ 6.0). A JMS Connector egy JCA szabványnak megfelelő Resource Adapter.




A J2CA standard architektúra egy Enterprise Information System (EIS) - akár lehet message provider is - és az alkalmazásszerver összekapcsolására. A csatlakozást resource Adapter-ekkel valósítja meg, melyek rendszer szintű szoftver meghajtók, és az alkalmazásszerver használ az EIS elérésére. Egy EIS lehet az üzenet-sorakoztató köztesalkalmazás is. A JCA megjelenése előtt nem volt szabványos eszköz a message-provider-ek csatlakoztatására. A resource adapterek akár menet közben cserélhető komponensek.

Egy ilyen architektúrában a következő szereplők léteznek: az alkalmazásszerver, mely a környezetet adja az alkalmazásnak, maga az alkalmazás, mely az üzleti funkciókat valósítja meg, valamint a EIS, mellyel az alkalmazás kommunikálna, illetve a resource adapter, melyen keresztül a kommunikáció megvalósul. J2EE környezetben az alkalmazás részeként Session Bean-ek és Message Driven Bean-ek kommunikálnak az EIS-szel, az előbbi a kimenő kommunikációért, az utóbbi a bejövő kommunikációért felelős. Az alkalmazásszerver és a resource adapter között un. szerződések vannak definiálva, melyek a következőket definiálják:

•Életciklus (indítás, leállás)

•Work unit management

•Message inflow (a resource adapter tudjon aszinkron üzeneteket küldeni az alkalmazásszerver felé - pl. MDB-nek)

•Transaction inflow

•Security management

•Connection management

A szolgáltatás szintjét emelendő a resource adapter a következő szolgáltatásokat nyújtja:

•Teljesítmény és erőforrás optimalizálás

•Kapcsolat pool-ozás

•MDB az üzenet mennyiségre érzékenyen reagál

•Dinamikus monitorozás és menedzsment

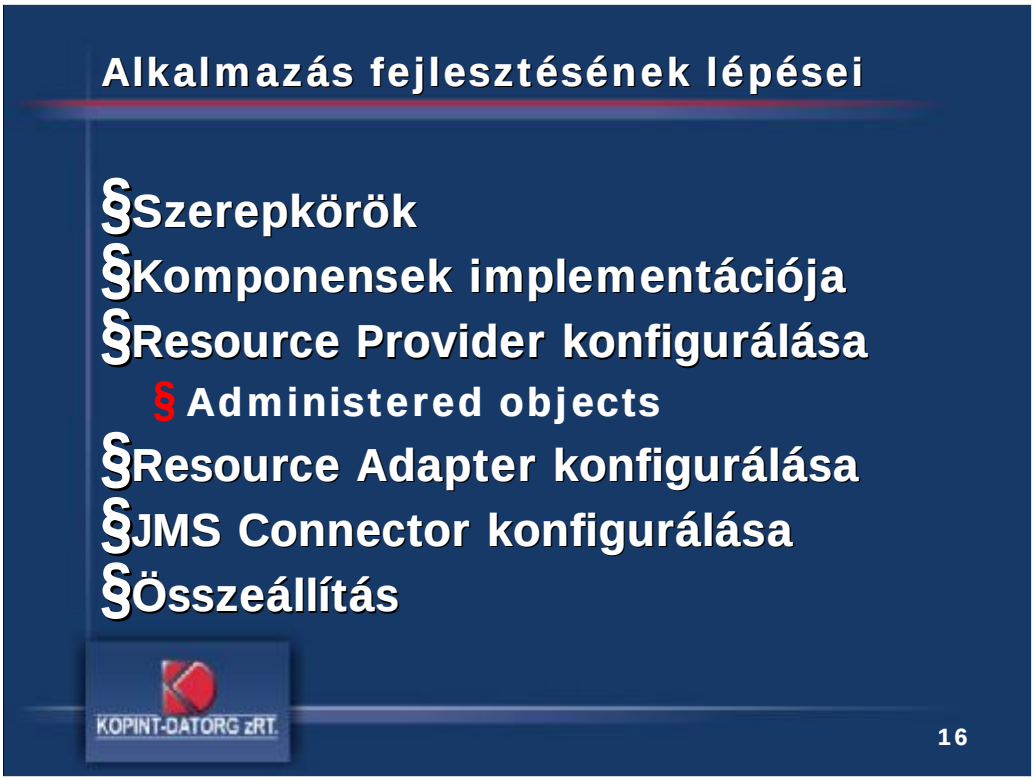
•Tranzakciókezelés

•Lazy evaulation

•Független az indítási sorrendtől

•Hiba loggolás és trace

Egy resource adapter kétféleképpen deployolható: globálisan, alkalmazásban.




J2EE alkalmazás fejlesztésekor, telepítésekor és üzemeltetésekor szét kell választani szerepköröket, melyek feladatai és felelősségei mások és mások. Persze egyazon személy több szerepkört is betölthet. A főbb szerepkörök a komponens készítő, mely felelős az alkalmazás komponenseinek (bean-ek, osztályok, stb.) programozásáért, előállításáért, az alkalmazás összeállító, aki összeállítja az alkalmazást egyetlen telepíthető állománnyá, és összeköti a komponenseket a telepítés leírók alkalmazásával, valamint a telepítő, aki feltelepíti az alkalmazást, és az aktuális környezetre szabja azt. Mindegyik szerepkörnek jól meghatározott feladata van, és metszetük minimális. A szerepkörök elhatárolódása megmutatkozik a fejlesztési folyamatban is.

Az alkalmazás megtervezése után előállnak a komponensek tervei is, melyeket a komponens készítő állít elő, használva a különböző API-kat, jelenleg a JMS API-t, amelynek objektumai korábban kifejtésre kerültek. Ezekben a komponensekben definiálnia kell különböző erőforrásokhoz tartozó logikai neveket, és a logikai nevekhez tartozó típusokat. Logikai neveket ConnectionFactory-khoz, valamint Queue-hoz és Topic-hoz (egységesen Destination) kell felvenni, melyet a forráskódban használunk. EJB 2.1 verziójú MDB esetén a telepítés leíró fájlban, míg a 3.0-ás EJB esetén a forrásban annotationként kell felvenni az erőforrásokat, és az erőforrásokat linkkel összekapcsolni a logikai nevekkel. Valamint itt lehet megadni az EJB viselkedését tranzakciókban. Telepítés leíróban kell megadni az alkalmazás által tartalmazott modulokat is. A legegyszerűbb esetében a Bean-eket tartalmazó EJB modult és a resource adaptert (ami jelen esetben a JMS Connector). A message provider üzemeltetőjének kell a konkrét RP ConnectionFactory-kat és Destination-okat létrehozni. Be kell állítani a különböző objektumok tulajdonságait és egy címtárban regisztrálni azokat. MQ esetén ez alapesetben lehet fájl, LDAP vagy WebSphere


Az alkalmazásszerver új monitor és menedzsment funkciói lehetővé teszik a gyors felismerést és reagálást a következő helyzetekben:

-Változó üzleti és felhasználói minták

-Váratlan terhelés

-Performance bottleneck

-Váratlan események

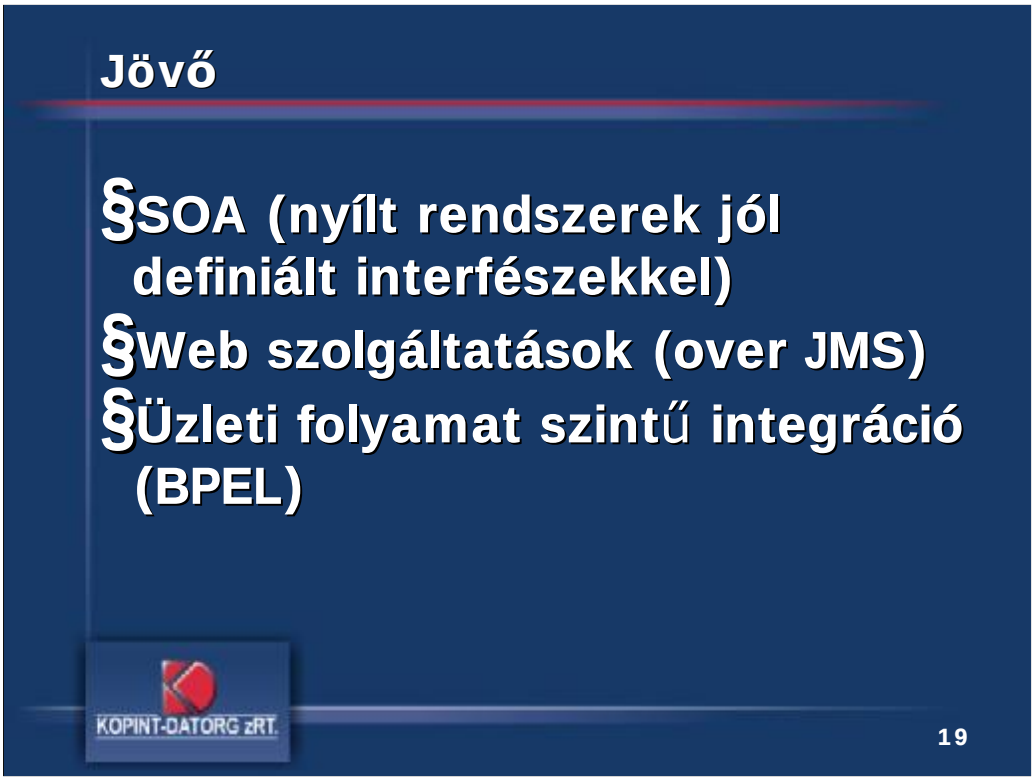
Az egyik legfontosabb újítás a JMX 1.2 megjelenése, mely a Java 1.5 része is, és lehetővé teszi az alkalmazások számára különböző menedzsment és monitor tulajdonságok lekérését, vagy akár beállítását is az alkalmazás újraindítása nélkül. Ezen kívül különböző operációkat is végrehajthatunk ezen a szabványos interfészen keresztül, valamint értesítéseket fogadhatunk. Az virtuális gép és a konténer is biztosít un. MBean-eket, melyek olyan Java objektumok, melyek adattagjait és metódusait JMX felületen keresztül el lehet érni, valamint a programozók is deklarálhatnak ilyeneket a saját alkalmazásukban.

Az alkalmazásszerver megvalósítja a J2EE management (JSR-77) specifikációt is, mely egy standard a J2EE alkalmazásszerverek menedzselésére. Így alkalmazásszerver független eszközöket lehet fejleszteni erre a feladatra.

Az alkalmazásszerver a J2EE application deployment (JSR 88) spacifikációt is megvalósítja, mellyel platformfüggetlen módon lehet összecsomagolt alkalmazásokat telepíteni, konfigurálni valamint eltávolítani. Az Oracle az alkalmazásszerveréhez alapban nyújt ilyen eszközöket. Ezek parancssori eszközök, Ant task-ok, valamint lehetőség nyílik különböző szkript nyelvek segítségével (Groovy, Beanshell, Jython) telepítő szkriptek összeállítására is. A JDeveloper fejlesztőeszköz is ezen a felületen csatlakozik az

# Demó

- Oracle JDeveloper
- Standalone OC4J
- EJB 3.0
- Összecsomagolás és telepítés
- Application Server Control
  - Alkalmazás (EJB, RA)
  - Statisztika
  - Napló
  - JNDI browser
  - MBeans browser
- Spy




A bemutatott alkalmazás az integráció viszonylag alacsony szintjén elhelyezkedő alkalmazás, két különálló rendszert köt össze pont-pont kapcsolattal. A jelenlegi trend azonban az alkalmazások egymástól való függetlenítése, és bizonyos alkalmazások csak szolgáltatásokat nyújtsanak, valamint más alkalmazások szolgáltatásait használják fel működésükhöz. Ezzel egymástól független alkalmazások fejleszthetők, és egy lazán kapcsolt rendszer fejleszthető ki, ahol a komponensek fekete dobozként viselkednek, csak az interfészük ismert. A SOA architektúra megvalósítására alkalmas eszköz a web szolgáltatások, ahol a HTTP(S) felületen lehet, szabványos módon (SOAP, WSDL, UDDI) lehet a különböző szolgáltatásokat és interfészeiket definiálni, valamint használatba venni. Ezekhez kitűnő eszközök is a rendelkezésünkre állnak, melyek az interfész alapján képesek kódot is generálni. A web szolgáltatások elnevezésében a web megtévesztő lehet, hiszen immár nem csak a gyakran használt HTTP(S) protokollon keresztül lehet szolgáltatásokat meghívni, hanem az átviteli protokoll tetszőlegesen cserélhető, így például lehet közönséges e-mail, de akár a bonyolult infrastruktúrát feltételező JMS is. Ennél is magasabb szinten épül a folyamat szintű integráció, ahol üzleti folyamatokat tervezzük, és a folyamatok csomópontjaiban használunk fel web szolgáltatásokat, így mintegy magasról koordináljuk a különböző alkalmazások együttműködését.