Debreceni Egyetem
Informatikai Kar

Szolgáltatás-orientált programozás az Oracle-ben

Témavezető:
Dr. Juhász István
egyetemi adjunktus

Készítette:
Ács László
programtervező-matematikus

Debrecen
2009

Tartalom

Köszönetnyilvánítás

Szeretnék köszönetet mondani Dr. Juhász István tanár Úrnak, aki elvállalta a diplomamunkám témavezetését, és tanácsaival segítette annak elkészítését.

Továbbá szeretnék köszönetet mondani családomnak és barátaimnak, akik minden lehetőséget megteremtettek ahhoz, hogy ez a dolgozat elkészülhessen.

I. Bevezetés

Napjaink üzleti informatikai világának egyik varázsszava a szolgáltatás és a szolgáltatás-orientált architektúra (SOA). A nagy cégek (és azok vezetői) szeretnek szolgáltatásokban gondolkozni, tehát szolgáltatásokat várnak el más cégektől, ezért a szoftverfejlesztő vállalatok is próbálnak szolgáltatásokat nyújtani.

Sokan (tévesen) azonosítják a szolgáltatás-orientált architektúrát (SOA) a webszolgáltatásokkal. Ha szigorúan vesszük, akkor a web szolgáltatások csak egy lehetséges megvalósítása a SOA-nak, illetve a SOA-hoz kapcsolódó ajánlásoknak.

Másrészt a vállalatok (különösen a nagyvállalatok) szeretik adataikat biztonságban tudni, adataikat lehetőleg minél biztonságosabban tárolni. Ezért jellemzően adatbázis-kezelő rendszereket alkalmaznak erre a célra. A nagyvállalati szinten a legelterjedtebb adatbázis-kezelő az Oracle vállalat által készített Oracle Database.

A szolgáltatás-orientált fejlesztéshez az Oracel egy teljes szoftvercsomagot kínál (Oracle SOA Suite), amely megpróbál minden olyan eszközt magában foglalni, amelyre szükség lehet egy szolgáltatásokra épülő vállalati szintű architektúra kiépítéséhez.

Ebben a dolgozatban szeretnék áttekintést nyújtani a szolgáltatás-orientáltságról, a webszolgáltatásokról, valamint arról, hogy ezen eszközök hogyan jelennek meg az Oracle által nyújtott komplex szoftverrendszerben.

II. SOA

A nagyvállalatok informatikai rendszerei igen összetettek, jellemzően több száz alkalmazásról, és az alkalmazások megvalósításához több tucat technológiáról beszélhetünk, amelyek együttesen szolgálják ki az üzleti igényeket. Ez leginkább abból eredeztethető, hogy az egyes vállalatok az évek (évtizedek) alatt nem valósítottak meg eléggé kézben tartott irányítást, illetve, sok esetben a már meglévő, de idejétmúlt technológiákra épülő alkalmazásokat használják, és nem vállalják az új technológia bevezetésének, és/vagy a program(ok) újraírásának költségeit.

Egy ilyen környezetben az egyik legkomolyabb probléma, hogy a funkciók, és adatok el vannak szórva az egyes alkalmazások között, gyakran ugyanaz a funkció több alkalmazásban is implementálva van. A gond az, hogy egy üzleti folyamat jellemzően nem egy alkalmazás használatát jelenti, hanem az alkalmazásokat közösen kell használni, az alkalmazásoknak egymásból kell információkat kinyerniük, egymással kell együttműködniük.

A szoftverfejlesztők célja az, hogy egy ilyen összetett környezetben is hatékony fejlesztést tudjanak megvalósítani úgy, hogy a meglévő eszközöket minél hatékonyabb használják fel. Ebből fakadóan olyan megoldást kell alkalmazni, amely a meglévő rendszerek minél nagyobb számával kompatibilis, azaz olyan szabványos megoldást kell alkalmazni, amelyet lehetőleg minél több gyártó támogat, és a lehetőségekhez mérten könnyen illeszthető a különböző technológiákhoz.

Fontos elvárás még, hogy hatékony legyen, azaz tegye lehetővé a gyors fejlesztést (az üzleti érték gyors teremtését). A gyors fejlesztés egyik alapfeltétele, hogy amit lehet, azt használjuk fel újra, ne kelljen ismét feltalálni a spanyolviaszt.

Az informatika felé állandó elvárás, hogy jól dokumentált munkákat készítsen, és jellemző követelmény még az informatikán belül, hogy az újrafelhasználható komponensek információi könnyen elérhetőek legyenek, azok interfészei egyértelműek legyenek, lehetőleg minimalizáljuk az interfész félreértelmezésének veszélyét.

A SOA (Service-Oriented Architecture) nem egy technológia a heterogén, elosztott rendszerekben jelentkező problémák megoldására, hanem egy gondolkodásmód. Maga a fogalom már

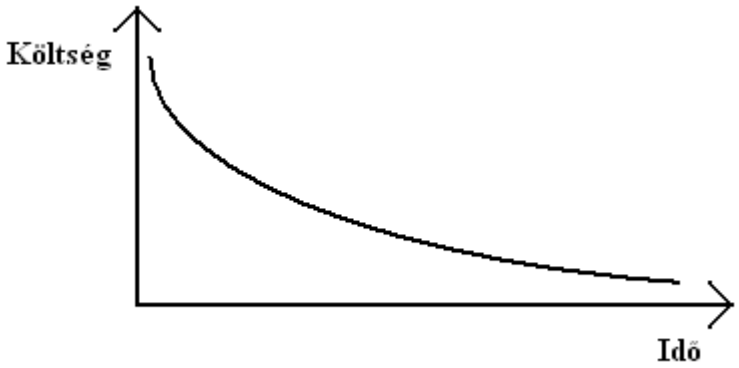
közel két évtizede létezik, de csak az utóbbi ~5 évben vált igazán elterjedtté. Talán a legfontosabb, amit a SOA-val kapcsolatban tudni kell, hogy alapvető célja a vállalat informatikai rendszereiben fellelhető „értékek” megkeresése, és ezek publikálása szolgáltatásként az ügyfelek felé. Ilyen értékek lehetnek: prezentációs szolgáltatás, üzleti folyamat (darab), üzleti adat, üzleti funkció, integráció, vagy bármi, amit az adott vállalat üzleti értéknek tekinthet.

A SOA rendszerfejlesztési elve az alábbi megközelítéseket ötvözi:

- **Moduláris programozás:** a funkciókat elkülönítjük egymástól, és nem komplett rendszereket fejlesztünk, hanem modulokat.
- **Újrafelhasználhatóság:** az egyes modulokat úgy kell megtervezni, hogy lehetőleg minél szélesebb felhasználási lehetőséget támogasson.
- **Lazán csatoltság:** a moduloknak nem szabad szorosan függniük egymástól, tehát az egyes felületek nem függhetnek attól, hogy kinek is fejlesztjük a modult (irreleváns-nak kell lennie, hogy kinél merült fel az igény először, a modulnak a lehető legáltalánosabb felületet kell nyújtania).
- **Különállóan kezelhető:** az egyes modulokat egymástól függetlenül lehessen üzemeltetni, adminisztrálni, skálázni.

A SOA célja a (nagy)vállalati rendszereken belül a több (al)rendszert érintő üzleti folyamatok strukturált megvalósításának segítése. Az egyes folyamatokat a rendszerek által felajánlott szolgáltatások összekapcsolásával építjük fel, tehát egy új folyamat implementálása részben a már meglévő szolgáltatások felhasználásával történik, így a már meglévő szolgáltatások kompozíciójából újabb, esetleg magasabb szintű szolgáltatásokat hozhatunk létre.

A SOA nagy ígéretének tekintik, hogy az egyes alkalmazások fejlesztési költsége az időben előre haladva a végtelenben a nullához közelít. Az induló költség magas, mivel egy új technológiát kell bevezetni, illetve ezzel a technológiával kell megjeleníttetni az eddig meg lévő alkalmazásokat, ami jellemzően csak évek múltán térül meg, de hosszú távon jobb eredmény nyújt.



1. ábra A költség és az idő viszonya a SOA-ban

Ezen kijelentés mellett szól, hogy az idő előre haladtával egyre több szolgáltatás fog a rendelkezésünkre állni, amelyek felhasználásával meg tudjuk valósítani a folyamatainkat. Ebből adódóan a fejlesztések egyre inkább eltolódnak abba az irányba, hogy csak szolgáltatásokat kell egymáshoz kötni, és már nem az egyes szolgáltatásokat kell kódolni.

Eddig is voltak újrafelhasználható eszközeink (eljárások, osztályok), de azok jellemzően sokkal kisebb egységet képviselnek, kevésbé összetettek, egy rendszerre korlátozódnak és nem, vagy csak nehézkesen biztosítják a rendszerek közötti átjárhatóságot.

A szolgáltatások interfésze egyfajta „szerződés” a szolgáltató, és az azt igénybe vevő ügyfél között, definiálja a híváshoz szükséges paramétereket, a visszatérési értékeket, és a hívás során előfordulható hibákat.

Az interfésszel szemben támasztott követelmények:

- **Egységes:** elnevezési konvenciókat követ, tehát a szolgáltatás nevéből már lehet következtetni a viselkedésére; az azonos fogalmakhoz azonos adattípusokat használ (pl.: a dátumokat ugyanolyan formátumban várja, az időt egy adott időzóna szerint, vagy az időzóna megjelölésével várja); illetve azonos programozási irányelveket követ.

- **Jól dokumentált:** Könnyen visszakereshető, illetve megállapítható egy szolgáltatásról, hogy mit kínál, és ehhez milyen elvárásai vannak.

- **Technológiailag is azonos:** Az egyes szolgáltatásokat lehetőleg ugyanazon technológiával készítsük, tegyük elérhetővé (ez jellemzően a webszolgáltatások).

III. Webszolgáltatások

A webszolgáltatások egy, a W3C által készített szabványos technológia a szolgáltatások ajánlására. Előnye, hogy minden gyártó támogatja.

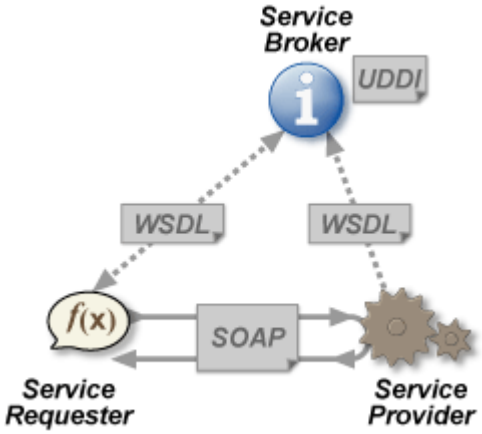
A szolgáltatásoknak is (mint minden szoftvernek) van életciklusa: a felmerült igényeket megvizsgálják, hogy érdemes-e implementálni, illetve, amennyiben erre a válasz igen, akkor érdemes-e közzétenni, vagy csak belső eszközként fogják használni. Ha a közzététel mellett döntenek, akkor definiálni kell a szolgáltatást, és meg kell tervezni. Ez után jön az implementáció, és a telepítés. Ha a közzététel mellett döntöttek, akkor publikálják a szolgáltatásukat. Ezt követi az üzemeltetés, és a felügyelet, tehát a szolgáltatás tényleges nyújtása mások számára. Amennyiben a szolgáltatás nem megfelelő, vagy készült egy újabb verzió belőle, tehát leváltásra, vagy eltávolításra kerül, akkor következik a kivezetési szakasz, amely a szolgáltatás életciklusának végét jelenti.

A webszolgáltatások legfontosabb elemei:

- **WSDL (Web Service Description Language):** egy szabványos interfész leíró nyelv.
- **SOAP (Simpla Object Access Protocol):** szabványos protokoll a szolgáltatások elérésére. Többféle protokoll felett működhet, jellemzően vagy SOAP over HTTP, vagy SOAP over JMS. A HTTP protokollon történő kommunikáció előnye, hogy a tűzfalak jellemzően nem blokkolják, tehát nem kell külön portot megnyitni a használatához.
- **UDDI (Universal Description, Discorvery and Integration):** szabvány a szolgáltatások regiszterekben történő publikálásához.

Egy szolgáltatás igénybevételének általános menete:

1. Meghatározzuk, hogy mire van szükségünk, mik az igényeink.
2. Egy szolgáltatás nyilvántartótól (service broker) WSDL-el keresünk egy olyan szolgáltatást, amely kielégíti az igényeinket.
3. Az így kapott szolgáltatótól SOAP használatával meghívjuk a felkínált szolgáltatást.



2. ábra a webszolgáltatások általános működése[1]

A fenti folyamatból a Service Borker akár ki is hagyható, amennyiben pontosan tudjuk, hogy mit akarunk meghívni.

1. WSDL

Miután az üzenetek formátumai és a kommunikációs protokollok szabványossá váltak a web közösségen belül, szükségessé (és lehetővé) vált a kommunikáció leírása strukturált módon. A WSDL (Web Services Description Language) ezt valósítja meg egy XML-re épülő nyelvtan megadásával. A WSDL szolgáltatás definíciók leírást biztosítanak az elosztott rendszereknek, és az alkalmazások közötti kommunikációs részletek automatizálására. 2007. június 26.-án W3C ajánlás lett.

Egy WSDL dokumentum a szolgáltatásokat, mint hálózati végpontok, vagy portok összességeként definiálja. Különválasztja a végpontok és üzenetek absztrakt definícióját a konkrét hálózati megvalósításuktól és a konkrétan használt adattípusoktól. Így lehetőség nyílik az absztrakt definíciók újrafelhasználására: az *üzenetek*, amelyek a kommunikáció során használt absztrakt adatok definíciója, illetve a *portok*, amelyek a műveletek absztrakt gyűjteményei. Egy WSDL dokumentum a szolgáltatások definiálására a következő elemeket használja:

- **Típusok (types):** egy tároló az adattípusok definiálására valamilyen típus rendszer felhasználásával (például XSD).

[1] Forrás: http://en.wikipedia.org/wiki/File:Webservices.png

- **Üzenet (message):** egy absztrakt, tipizált definíciója a kommunikációban használt adatnak.
- **Művelet (operation):** a szolgáltatás által nyújtott művelet absztrakt definíciója.
- **Port típus (port type):** egy vagy több végpont által nyújtott műveletek absztrakt halmaza.
- **Hozzárendelés (binding):** egy konkrét protokoll és adattípus definíció egy port típus számára.
- **Port:** egy önálló végpont, amelyet egy hozzárendeléssel és egy hálózati címmel adunk meg.
- **Szolgáltatás (service):** kapcsolódó végpontok egy halmaza.

A WSDL nem definiál egy saját típus leíró nyelvet, mivel nem lehet elvárni egy leíró nyelvtől sem, hogy alkalmas legyen a világon található összes adat ábrázolására. Mindazonáltal, mivel szükség van egy széleskörű eszközkészlettel rendelkező adatleíró nyelvre, ezért alapértelmezésben támogatja az XML Schema specifikációt (XSD), de lehetőség van más leíró nyelvek használatára is az általánosság megtartásának érdekében.

2. SOAP

A SOAP (Simple Object Access Protocol) a webszolgáltatások közötti kommunikáció protokollja, amellyel szabványos, XML alapú üzenetküldés valósítható meg; nyelv és platform független, illetve különböző protokollok felett képes működni (pl.: HTTP). Üzeneteinek szintaktikáját a WSDL írja le.

A HTTP és XML közös használatának előnye, hogy eme két technológia a rendszerek nagyon széles köre által támogatott.

A HTTP mellett szól, hogy átjut a tűzfalakon és proxikon, ugyanakkor, ha nem használunk ESB-t, akkor a szerepek rögzítettek, és csak a kliens tudja használni a másik szolgáltatásait.

Az XML használatának megvannak az előnyei és hátrányai is. Előnye, hogy emberek is könnyen tudják olvasni, könnyű benne a hibakeresés, és megoldja a rendszerek közötti különbségekből fakadó problémákat (pl.: byte sorrend), ugyanakkor lassítja a feldolgozási sebességet, különösen nagy és bonyolult objektumok esetén. A CORBA például egy jóval rövidebb, bináris formátumot használ. Gyorsításra ad lehetőséget a bináris XML használata.

3. UDDI

Az UDDI (Universal Description, Discovery and Integration) egy platform független, XML alapú szabvány a szolgáltatások közzétételére az interneten. Az UDDI egy címtár, amely információkat tartalmaz a szolgáltatásokkal kapcsolatban, illetve a szolgáltatások interfészeinek a leírását tárolja WSDL-el megadva; a kommunikációhoz a SOAP protokollt alkalmazza.

Három részből tevődik össze:

- **Fehér lapok:** cím, elérés és ismert azonosítók („telefonkönyv”).
- **Sárga lapok:** ipari besorolás általános taxonómiák alapján („szaknévsor”).
- **Zöld lapok:** technikai információk a közzétett szolgáltatásokról.

Az eredeti elképzelés az volt, hogy bárki, akinek szüksége van egy szolgáltatásra, csak megkeres egy szolgáltatás nyilvántartót (service broker / registry), és ott keres egy szolgáltatást, amely megfelel a feltételeinek.

Ebben az esetben a nyilvántartó kritikus pont mindenki számára. A felhasználók számára azért, mert csak a nyilvános szolgáltatások látszanak feléjük, a szolgáltatók felé pedig azért, mert egy jó pozíció megszerzése lenne a kritikus feltétel (a keresésekre adott válasz listákban az Ő szolgáltatásuk lehetőleg minél előkelőbb helyet foglaljon el).

4. ESB

A szolgáltatásoknál fellépő egyik probléma, hogy ha az egyes alkalmazások közvetlenül címzik meg a szolgáltatásokat, akkor pont-pont kapcsolatok alakulnak ki, amelyek nehezen karbantarthatóak, különösen akkor, ha az egyes alkalmazások több szolgáltatással is kapcsolatban állnak.

Az ESB (Enterprise Service Bus) erre a problémára nyújt megoldást, egy köztes alkalmazásként jelenik meg a szolgáltatások és az azokat felhasználó alkalmazások között. Mind a szolgáltatások, mind az alkalmazások közvetlenül csak az ESB-vel kommunikálnak, egymással csak közvetve.

Fő előnye, hogy lehetőséget ad a központi karbantartásra, tehát a szolgáltatásokat, illetve azok kapcsolatait egységesen tudjuk kezelni. Lehetséges vele a monitorozás, és megfelelő megvalósítás esetén jól paraméterezhető. Biztosítja a szolgáltatást igénybe vevőjének (az alkalmazás, vagy egy másik szolgáltatás) és a szolgáltatás laza csatolását, mivel egyik sem tud közvetlenül a másikról.

Fontos szempont, hogy elrejti a technikai részleteket a szolgáltatást igénybevevők elől. A hívó alkalmazásnak nem kell tudnia, hogy pontosan hol helyezkedik el a szolgáltatás, a hívó csak az ESB-vel kommunikál, és az ESB megoldja, hogy a kérések pontosan azokhoz a végpontokhoz kerüljenek, amelyek képesek teljesíteni őket. Ha szükséges, akkor üzenet- és protokoll transzformációt hajt végre, üzenetirányításra is képes (routing), megoldhatóak vele a biztonsági kérdések. Képes beleolvasni az üzenetekbe, és az így kapott információk alapján továbbítja azt a megfelelő kiszolgálóhoz, vagy kiegészítheti az üzeneteket. Ha szükséges, akkor üzleti logika is kódolható bele, bár ez ellenjavallt, mivel az üzleti logika lekódolására a szolgáltatások valóak.

Lehetőséget ad a hívási paraméterek és/vagy a visszatérési értékek transzformációjára a hívó és a szolgáltatás követelményeinek megfelelően.

5. Service Registry

A szolgáltatásokat leíró meta adatok tárolását teszi lehetővé az UDDI szabványnak megfelelően. Előnye, hogy a szolgáltatás leírások egységesek, könnyen visszakereshetőek, és a kötött leírási mód fegyelmet követel meg a vállalaton belül. Értesítési mechanizmusokkal biztosítja az érintettek változásokról való értesülését, valamit támogatást nyújt a szolgáltatások életciklusának kezelésében.

IV. Oracle SOA Suite

Az Oracle SOA Suite alkalmazások egy olyan gyűjteménye, amely támogatja a szolgáltatások létrehozását, telepítését és felügyeletét. Lehetővé teszi, hogy a szolgáltatásokat létrehozzuk, kezeljük és összekomponáljuk őket alkalmazásokká és új, összetettebb szolgáltatásokká.

Támogatást nyújt mind a programozóknak, mind az üzleti elemzőknek, hogy minél hatékonyabb munkát tudjanak végezni. Segítségével a szolgáltatások szintjén az üzleti elemzők is képesek folyamatokat definiálni grafikus környezettel, valamint lehetőségük van a meglévő folyamatok, és szabályok változtatására akár futási időben is.

Betekintést nyújt a vállalat pillanatnyi állapotába, ezzel segítve a magas szintű döntéshozatalt, illetve a gyors és hatékony hibaelhárítást, valamint, ami sokkal fontosabb, a hiba megelőzést (pl.: monitorozhatjuk a készleteinket, és amennyiben ezek egy bizonyos szint alá esnek, akkor beavatkozhatunk).

Támogatja az automatizált folyamatok közben fellépő emberi beavatkozást (jellemzően jóváhagyást) igénylő folyamatok hatékony kezelését.

Az Oracle SOA Suite a következők fő elemekből áll össze:

- **ISE (Integrated Service Environment)** egy fejlesztői környezet a szolgáltatások hatékony fejlesztéséhez
- **Oracle BPEL Process Manager** a szolgáltatások üzleti folyamatba történő egybefoglalásához
- **ESB** a már létező IT rendszerek és üzleti partnerek összekapcsolásához
- **Oracle Business Rules**, amelyet az üzleti felhasználók / elemzők is tudnak használni, és dinamikus döntéseket lehet vele hozni akár futási időben is
- **OracleAS Integration Business Activity Monitoring**, amellyel meg lehet figyelni a folyamatokat és a különálló eseményeket, valamint valós idejű betekintést nyújt a vállalat helyzetébe, az üzleti folyamatokba és a rendszerekbe.

- **Oracle Web Service Manager**, amellyel megvalósíthatóak a hitelesítési és titkosítási elvek biztosítása és kezelése

- **UDDI Registry**, amellyel publikálhatjuk és kezelhetjük a webszolgáltatásokat.

Az Oracle SOA Suite részei:

<table>
<tr><th>Integrated Services Environment</th><th>Oracle BPEL Process Manager</th><th>Oracle Enterprise Servise Bus</th><th>Oracle Business Rules</th><th>Oracle Busi-ness Activity Monitoring</th><th>Oracle Web Serives Manager</th></tr>
<tr><td>Oracle JDeveloper</td><td>Native BPEL</td><td>Adapters</td><td>Decision Service</td><td>Analytics</td><td rowspan="4">Security</td></tr>
<tr><td>Application Development Framework</td><td rowspan="2">Human Workflow</td><td>XSLT Transform</td><td rowspan="2">Rule Author</td><td>Events</td></tr>
<tr><td>Oracle TopLink</td><td>Routing</td><td>Monitoring</td></tr>
<tr><td rowspan="2"></td><td colspan="4">Oracle Application Server, J2EE, WS-*, Event Services</td></tr>
<tr><td colspan="2">B2B</td><td colspan="2">UDDI Registry</td></tr>
</table>

1. Integrated Service Environment (ISE)

Az Oracle JDeveloper a fejlesztői eszköze az Oracle SOA Suite-nak, amellyel szolgáltatásokat készíthetünk, nyilvántarthatjuk őket, illetve összehangolhatjuk őket üzleti folyamatokká.

Lehetővé teszi és támogatja a fejlesztők számára a szolgáltatásokon alapuló összetett alkalmazások tervezését, létrehozását, felderítését, tesztelését, telepítését. Ugyanúgy támogatja a SOA alapelveket és XML alap szolgáltatások szabványát, mint a szokásos Java, J2EE és PL/SQL komponensek és modulok fejlesztését.

Az Oracle ADF (Application Development Framework) egy modell-orientált SOA keretrendszer, amely automatizálja és felügyeli a folyamatokat és adatszolgáltatásokat, valamint egy szabványos adat és szolgáltatás megfeleltetési réteget nyújt.

Az Oracle TopLink segítségével relációs és XML adatokhoz férhetünk hozzá. Grafikus eszközöket nyújt a relációs és XML adatok objektumokkal történő megfeleltetéséhez.

2. Oracle BPEL Process Manager

Célja egy olyan keretrendszer nyújtása, amely segíti a folyamatok tervezését, adminisztrálását és monitorozását a BPEL standardoknak megfelelően. A következő tulajdonságokkal rendelkezik:

- A webszolgáltatásban használatos standardok alkalmazása (XML, SOAP és WSDL).
- Dehidratáció: engedélyezi a hosszan tartó folyamatok állapotának automatikus adatbázisban történő tárolását
- Támogatja a feladatok párhuzamos végrehajtását
- Hiba- és kivételkezelés tervezési és futási időben
- Esemény időtúllépések (timeout) és értesítések
- A folyamatok skálázhatósága és megbízhatósága
- A folyamatok kezelése és adminisztrálása
- Verziókezelés

Webszolgáltatások meghívása: Egy webszolgáltatást szinkron és aszinkron módon hívhatunk. A szinkron hívások közel azonnali választ feltételeznek, és blokkolják a BPEL folyamat végrehajtását. Az aszinkron hívások nem blokkolják a végrehajtás, és különösen hasznosak akkor, amikor a kérés feldolgozása hosszabb időt is igénybe vehet.

Mindkét típusú híváshoz szükség van a következő elemekre: *PartnerLink*[2] és *Invoke*.

Egy aszinkron szolgáltatás esetén még szükséges egy *Receive* és egy Correlation ID (biztosítja, hogy a kapott válasz az általunk küldött kéréshez tartozik. A PartnerLink-re nézve egyedi, és a szolgáltatásnak is elküldi).

[2] Az egyes BPEL műveleteket dőlt betűvel jelölöm a továbbiakban.

Tehát egy webszolgáltatás meghívásához a következő műveleteket kell a BPEL folyamathoz adni:

- Egy *PartnerLink*-et, amellyel megadjuk azt a külső szolgáltatást, amellyel a BPEL folyamatunk kapcsolatba lép.
- Egy *Scope*-ot, hogy egyesítsük az összetartozó műveleteket, amelyeknek így lehetnek saját lokális változóik, és hibakezelőik.
- Egy *Invoke* műveletet, amellyel megadjuk, hogy a szolgáltatás mely műveletét akarjuk meghívni.
- Két *Assign* műveletet. Az első tartalmazza a bemeneti adatokat, és ezeket adjuk át a szolgáltatásnak, a második visszaadja a szolgáltatástól kapott választ.

Aszinkron hívás esetén még hozzá kell adni egy *Receive* műveletet is az *Invoke* után.

Hibakezelés: lehetővé teszi a BPEL folyamatok számára a külső webszolgáltatásokból kapott hibaüzenetek vagy egyéb kivételek kezelését, és a hibaüzenetek generálását, amennyiben üzleti vagy futási hiba lép fel. A hibákat két csoportba oszthatjuk:

- **Üzleti hibák:** ezek az alkalmazás-specifikus hibák akkor lépnek fel, ha egy *Invoke* művelet hibát kap válaszként, vagy amikor egy alkalmazás végrehajt egy *Throw* műveletet. Az üzleti hibák az információval kapcsolatos problémák eredményei, például amikor egy keresett azonosítót nem lelünk az aktuális adatbázisban.
- **Futási hibák:** ezek a hibák akkor keletkeznek, amikor egy BPEL folyamat hibásan próbál meg használni egy értéket, vagy logikai hiba van, például egy végtelen ciklus. A futási hibák a BPEL folyamaton, vagy magukon a webszolgáltatásokon belüli hibák eredményei.

Az egyes *Scope*-okhoz lehetnek catch vagy catchAll részek, amelyek elkapnak egy meghatározott típusú hibát, vagy minden, addig el nem fogott hibát.

A BPEL folyamaton belüli hiba jelzésére használhatjuk a *Throw* műveletet. Amikor egy *Throw* műveletet adunk a folyamathoz, akkor ez automatikusan tartalmaz egy másolási sza-

bályt, amely a hiba nevét és típusát átmásolja a kimenetbe. A hiba, amelyet a művelet generál, a BPEL folyamaton belülinek tekintendő. Nem lehet egy aszinkron folyamaton felhasználni arra, hogy a klienssel kommunikáljunk. Három része van: name, faultName és faultVariable. A BPEL folyamat tud hibát küldeni más alkalmazásoknak, hogy jelezze a nem megfelelő működést.

Egy lehetséges hibakezelő kód XML-ben:

```
<faultHandlers>
 <catch faultName="client:ProjectRegisterFault"
 faultVariable="ProjectRegisterFault">
 <sequence name="Sequence_4">
  <scope name="SetProjectStatus">
   <variables>
    <variable name="projectStatusRequest"
     messageType="ns2:ProjectCollection_msg"/>
    </variables>
    <sequence name="Sequence_5">
     <assign name="AssignProjectStatus">
      <copy>
       <from variable="inputVariable" part="payload"
        query="/client:SOAProjectRegisterProcessRequest/ns4:RegProject/ns4:ID"/>
       <to variable="projectStatusRequest" part="ProjectCollection"
        query="/ns9:ProjectCollection/ns9:Projects/ns9:projid"/>
      </copy>
      <copy>
       <from expression="string('failed')"/>
       <to variable="projectStatusRequest" part="ProjectCollection"
         query="/ns9:ProjectCollection/ns9:Projects/ns9:status"/>
      </copy>
      <copy>
       <from variable="ProjectRegisterFault" part="payload"
        query="/client:SOAProjectRegisterProcessFault/client:status"/>
       <to variable="projectStatusRequest" part="ProjectCollection"
        query="/ns9:ProjectCollection/ns9:Projects/ns9:comment"/>
      </copy>
     </assign>
     <invoke name="SetFaultedProjectStatus" partnerLink="ProjectStatus"
      portType="ns2:ProjectStatus_ptt" operation="update"
      inputVariable="projectStatusRequest"/>
    </sequence>
   </scope>
  </sequence>
 </catch>
</faultHandlers>
```

Szenzorok: figyelik a BPEL műveleteket, változókat és a hibákat a futás közben. A következő típusú szenzorokat lehet definiálni a JDeveloper BPEL Designer-e segítségével, vagy kézzel, ha mi írjuk a szenzorok konfigurációs állományait:

- **Műveleti szenzorok:** a műveletek végrehajtását monitorozza a BPEL folyamaton belül. Például, vizsgálhatjuk, hogy mennyi időbe telik egy *Scope*, vagy éppen egy adott művelet végrehajtása.

- **Változó szenzorok:** a változók, vagy azok egyes részeinek monitorozására valóak. Például, megvizsgálhatjuk az egyes változók értékét egy BPEL folyamatra nézve annak kezdetekor és végekor.

- **Hiba szenzorok:** A BPEL folyamat során fellépő hibák monitorozására használható.

Amikor szenzorokat hozunk létre a JDeveloper BPEL Designer-ében, akkor két új állomány jön létre: sensor.xml (a BPEL folyamat által tartalmazott szenzorok definíciói) és sensorAction.xml (a szenzorok által végrehajtott műveletek definíciói). Ha szenzorokat definiálunk, akkor definiálnunk kell hozzájuk a megfelelő műveleteket is, hogy a szenzor által gyűjtött adatok melyik végponthoz kerüljenek.

Egy szenzor művelet számára a következő adatok szükségesek: név, a publikálás típusa (azt a célpontot reprezentálja, ahova a szenzor adatai kerülnek, ez lehet egy adatbázis, egy Java osztály, stb.), és azon szenzorok listája, amelyekre a művelet vonatkozik.

BPEL műveletek

A szabványos BPEL műveleteket két nagy csoportba lehet besorolni: alapműveletek, és strukturált műveletek. Az alapműveletek a legegyszerűbb módjai a külvilággal való kapcsolattartásnak, kölcsönhatásnak. Nem szekvenciális, egyedi lépések, amelyek kölcsön hatnak egy szolgáltatással, változtatják az átadott adatot, vagy hibákat kezelnek.

Webszolgáltatás interakciók: egy folyamat három műveleten keresztül tud kapcsolatba lépni a külvilággal, ezek: *Invoke, Reply* és *Receive*. Mindegyikük azonosítja azt a webszolgáltatás hívást, amelyhez tartoznak a következőek megadásával: a port típus, a művelet és a partner.

Az *Invoke* művelet segítségével tud a folyamat meghívni mások által nyújtott webszolgáltatásokat. A port típuson, a műveleten és a partneren felül még vannak input és output konténerek, a művelet számára kimenő, és a művelet által válaszként nyújtott adatok

számára. Egy ilyen művelet lehet szinkron vagy aszinkron is. Utóbbi esetben csak az input konténer szükséges.

A szolgáltatásokat az üzleti folyamatok a *Receive* és *Reply* műveleteken keresztül nyújtják. A *Receive* jelenti a szolgáltatás által nyújtott WSDL műveletet bemenetét. Ha a folyamatnak választ kell küldenie a hívó félnek, akkor egy *Reply* műveletre van szükségünk. Több *Reply* műveletet is definiálhatunk a folyamaton belül, de ezek közül csak egy válhat aktívvá a folyamat egy adott futása során.

A folyamat életciklusát befolyásoló műveletek: az üzleti folyamat egy példánya akkor jön létre, ha kap egy *Receive* vagy *Pick* műveletnek küldött üzenetet. Ezek a műveletek a createInstance attribútum igazra állításával jelzik, hogy egy új példányt kell létrehozni. A Pick művelet tartalmaz egy onMessage tagot, amely igazra állítja ugyanazt az attribútumot. Az első *Receive* művelet hatására létrejön a folyamat (ha nem volt előtte *Pick* művelet), az ez után lévő további *Receive* műveletek szokványos műveletekként kerülnek értelmezésre az adott példányon belül.

Az egyes példányok megkülönböztetésére a korrelációt (correlation) használja. Ezt használja fel arra, hogy eldöntse, melyik bejövő üzenet melyik példányhoz tartozik. Amikor egy üzenet érkezik, akkor megnézi, hogy van-e hozzá létező korreláció. Ha talál ilyet, akkor ahhoz küldi. Egyébként létrehoz egy újat a bejövő üzenet művelet, portType és partner információi alapján.

Adatkezelés: az *Assing* művelet segítéségével lehet egyik tárolóból a másikba másolni az adatokat, valamint új adatot létrehozni és beszúrni az adott kifejezésnek megfelelően. A kifejezések használata szükségszerű, ha a folyamatunk számára alapvető műveleteket akarunk végrehajtani, például növelni akarunk egy számlálót.

Minden *Assing* művelet egy vagy több <copy> tagot tartalmaz, amelyek pontosan egy <from> és egy <to> tagot tartalmaznak. Ezen elemeknek sok különböző alakjuk lehet, de ami fontos, hogy a forrásnak és a célnak típusban kompatibilisnek kell lenniük.

A <from> egyik legegyszerűbb alakja esetén csak egy tárolót adunk meg, ez azt jelenti, hogy a teljes tároló tartalma le lesz másolva. Ebben az esetben a célnak is egy tárolónak kell lennie.

Ennél egy kicsivel összetettebb eset, amikor a <from> egy tárolót és annak egy elemét jelöli meg. Ilyenkor a célnak szintén egy tárolót, és annak egy elemét kell jelölnie.

Egy további lehetőség, amikor a <from> egy XPath által kiértékelhető kifejezés (bármi lehet, amit az XPath elfogad). Ilyenkor a célnak egy konténernek és annak egy elemének kell lennie.

Tehát az üzenetek felül tudnak írni más üzeneteket, valamint az üzenetek részei is felül tudnak írni más üzenetek részeit.

Mivel az XPath-ra épít a BPEL, ezért kiterjeszti az XPath függvénykészletét. Ezek a függvények a szabványos BPEL névtérben vannak definiálva (http://schemas.xmlsoap.org/ws/2002/07/business-process/), és a bpws előtag van a névtérhez rendelve.

Egyéb alapműveletek: hibákat a *Throw* művelet segítségével jelezhetünk. Azért, hogy el tudjuk kapni, és kezelni tudjuk a hibákat, a hibáknak globálisan egyedi QName szükséges. Egy opcionális tárolót is megadhatunk, hogy jelezzük, hol vannak a hibával kapcsolatban lévő adatok. Például, egy *Throw* művelettel jelezhetünk egy hibát, majd egy *Reply* művelettel informálhatjuk a hívót a hibáról. A válasz felhasználhatja a hiba által jelzett tárolót, hogy pontosabb információt tudjon nyújtani.

A *Terminate* művelettel jelezhetjük, hogy a folyamatot azonnal le kell állítani, és minden futó részét meg kell szakítani.

A *Wait* művelettel jelezhetjük a folyamatnak, hogy várjon egy adott ideig, vagy egy adott időpontig.

Az *Empty* művelet nem csinál semmit. Akkor lehet hasznos, ha el akarunk kapni és kezelni akarunk egy hibát.

Strukturált műveletek: megadják a folyamaton belüli műveletek sorrendjét. Meghatározzák, hogy hogyan jön létre egy üzleti folyamat a különálló műveletekből.

A *Sequence* művelet egy vagy több műveletet tartalmaz, amelyeket folytonosan hajtunk végre. A műveleteket a *Sequence* műveleteken belüli megjelenésük sorrendjében hajtja végre.

Amikor a *Sequence* utolsó művelete is befejeződött, akkor maga a *Sequence* művelet is véget ér.

A *Switch* művelet a switch-nek felel meg a hagyományos programozási nyelvekből. Egy <case> ágakból álló rendezett listát tartalmaz, amelyet egy opcionális <otherwise> ág követhet. Minden <case> ág egy logikai XPath kifejezést jelent, amelyek a megjelenésük sorrendjében értékelődnek ki. Az első olyan <case> ágban található művelet hajtódik végre, amelynek a kifejezése igaz értéket ad. Ha a <case> ágak mindegyike hamis értéket ad, akkor az <otherwise> ág hajtódik végre. Ha nem adtunk meg <otherwise> ágat, akkor egy *Empty* műveletet tartalmazó jön létre automatikusan. Amikor a kiválasztott ág befejeződik, akkor a *Switch* művelet is befejeződik.

A *While* művelet mindaddig ismétli a tartalmazott műveletét, ameddig a megadott logikai XPath kifejezés hamissá nem válik.

A *Pick* művelet eseménykezelők egy halmazát tartalmazza, ahol mindegyikhez tartozik egy művelet. A műveletek akkor hajtódnak végre, ha a *Pick* elkezdődött, és az eseménykezelőhöz tartozó esemény bekövetkezett. Az eseményjelzők lehetnek időzítők, amelyek egy időtartam leteltét, vagy egy időpont elérését jelzik, vagy üzenetkezelők (onMessage), amelyek egy megadott partner, portType és operation hármastól várnak üzenetet. Az üzenetkezelők létrehozhatnak egy új példányt, ahogyan a *Receive* is létrehozhat. Csak az az eseménykezelő fog érvényesülni, amelyiknek elsőnek történik meg az eseménye.

A *Scope* művelet segítségével logikai egységekre bonthatjuk a folyamatot. Továbbá lehetővé teszi a hibakezelést egy logikai egységre nézve.

A *Flow* segítségével műveleteket hajthatunk végre párhuzamosan. Tetszőleges számú műveletet tartalmazhat. A *Flow* indulásakor minden része elindul, és akkor fejeződik be, amikor minden része befejeződött.

3. Oracle ESB

Legfőbb feladata az adatok mozgatása a végpontok között, legyenek azok a vállalaton belül vagy kívül. Nyílt szabványokat használ az eltérő alkalmazásokhoz történő csatlakozáshoz, illetve közöttük az üzleti dokumentumok irányításához és átalakításához. Segítségével moni-

torozhatjuk és felügyelhetjük az adatokat. Az ESB szolgáltatja az alap infrastruktúrát egy szolgáltatás- és esemény-alapú architektúrához.

Mint a szolgáltatások és események központi kapcsoló eleme, egy lazán kapcsolt keretrendszert nyújt, amely a vállalatok számára megnövekedett rugalmasságot és újrafelhasználhatóságot jelent, valamint lehetővé teszi a heterogén, üzenet-orientált rendszerek építését ipari szabványok alapján.

Az ESB tulajdonságai, amelyek segítik az alkalmazások integrálását, három fő csoportba sorolhatóak:

- **Csatlakoztathatóság:** adapter szolgáltatások, és SOAP hívások segítségével biztosítja. Az Oracle ESB eszközöket és varázslókat biztosít a szolgáltatások létrehozásához és meghívásához.
- **Dokumentum transzformáció:** segíti az egyik XML sémából a másikba történő transzformációt, ezáltal lehetővé teszi az adatcserét különböző alkalmazások között is.
- **Irányítás:** az XML állományokban tárol adat a forrástól a célig az irányítási (routing) szolgáltatásokon keresztül jut el. Meghatározza, hogy egy üzenet hogyan kerül továbbításra az egyes ESB-n belüli pontok között. Az irányítási szabályoknak mindenképp tartalmazniuk kell azon szolgáltatások halmazát, amelyeket meg kell hívni, amikor egy üzenetet kezel. Ezen felül, az irányítási szabályok beállításánál lehetőség van a következők megadására:
    - Legyen-e szűrési feltétel, amely esetén az üzenetek meta-adatait elemzi, mielőtt meghívja a szolgáltatásokat.
    - Legyen-e engedélyezve a dokumentum transzformáció.
    - Szinkron vagy aszinkron a végrehajtás.
    - A prioritási szint a többi szabályhoz képest.
    - Azon ESB rendszerek, amelyektől üzeneteket fogadhat el.

Irányítási szolgáltatás: kulcsfontosságú része az üzenetek mozgatásának az ESB-n belül (a belépési pontjától a kilépési pontjáig). Egy irányítási szolgáltatás a következő részekből áll össze:

- Egy WSDL állomány.
- Cél szolgáltatások és műveletek.
- Transzformációs definíciók.
- Szűrő feltételek.
- Végrehajtási mód (szinkron vagy aszinkron).
- Az ESB rendszerek, ahonnan elfogad üzeneteket.

A WSDL állomány megadja, hogy más szolgáltatások hogyan hívják az irányítási szolgáltatást (legyenek akár a vállalaton belüli vagy kívüli szolgáltatások). A többi rész megadja, hogy hova továbbítja az üzeneteket, hogyan küldi el azokat, és milyen változtatásokat végez az üzeneten (ha változtat egyáltalán), mielőtt továbbítja.

Domain-value Map: azok az alkalmazások, amelyeket együtt akarunk használni egy ESB-n belül, jellemzően különböző módon ábrázolják ugyanazt az információt. Az egyik alkalmazás például ábrázolhatja az országok nevét teljes név megadással, míg egy másik csak rövidítéseket használ. Egy domain-value map lehetővé teszi, hogy az egyik alkalmazásban lévő értékeket összerendeljük egy másik alkalmazásban lévő értékekkel.

Minden ilyen leképezés jellemzően az értékek egy speciális kategóriáját ábrázolja az alkalmazások között.

Miután definiáltunk egy ilyen leképezést, a fejlesztők ezeket fel tudják használni az adat transzformációt leíró állományaik készítéséhez, így az alkalmazás-specifikus értékeket futási időben is tudják használni.

Adapterek: Az Oracle ESB-hez nagyon sok konkrét adapter található, amelyek biztosítják a kétirányú átjárhatóságot az ESB és az egyes alkalmazások / technológiák között. Az adapterek az ESB felé szolgáltatásokként jelennek meg. 4 fő típusba sorolhatóak:

- **Alkalmazások:** egyes konkrét alkalmazásokhoz nyújt közvetlen kommunikációs kapcsolódási lehetőséget, például: Lotus Notes, Microsoft CRM, stb.

- **Adatbázisok:** egyes programok adatbázisokon keresztül kommunikálnak, tehát az eredményeiket, illetve paramétereiket bizonyos adattáblákba helyezik, vagy onnan várják. Az eme programokkal történő kapcsolattartásra alkalmasak az adatbázis adapterek.

- **Legacy:** alapvetően már nem használt, de egyes helyeken még meglévő alkalmazásokkal történő kommunikációt segítik ezek az adapterek.

- **Technológiai:** valamilyen konkrét technológiához nyújt kapcsolódási felületet. Például egyes alkalmazások meghatározott könyvtárakba helyezik le az eredményeiket állományok formájában, illetve a bemeneti értékeiket is állományokként várják. A velük való kommunikációra alkalmazhatóak ezek az adapterek.

Legrosszabb esetben egyedi adapterre van szükség, melyet nekünk kell megvalósítani.

4. Oracle Business Rules

A Business Rules Group[3] szerint, kétféle módon lehet definiálni az üzleti szabály fogalmát:

a) Egy üzleti elemző szemszögéből: egy üzleti szabály egy útmutatás, amely érinti az irányítást, az eljárásokat és a gyakorlatokat, vagy egy eljárás egy meghatározott tevékenységen vagy területen belül.

b) Egy információs rendszer szemszögéből: egy üzleti szabály egy megállapítás, amely meghatározza vagy tartalmazza az üzlet néhány jellegzetességét. Célja az, hogy meghatározza az üzleti struktúrákat, irányítsa, vagy befolyásolja az üzlet működését.

Egy, az üzleti szabályokat támogató terméknek tartalmaznia kell egy dekleratív szabályleíró nyelvet, egy hatékony következtető motort, és eszközöket, amelyek támogatják a szabályok definiálását és kezelését. Lehetővé kell tennie a fent említett két megközelítés közötti szakadék áthidalását.

[3] Business Rules Group, http://www.businessrulesgroup.org

A Business Rules-t akkor érdemes használni, ha az alábbi esetek valamelyik fennáll:

- A vállalat számára nagyobb agilitás szükséges az üzleti folyamataiban. Jellemzően gyorsan szeretnének reagálni a versenyhelyzetek illetve a szabályozások változására, és a folyamatok működését azok módosítása és újratelepítése nélkül szeretnék megoldani.

- Az üzleti folyamatokat dinamikusan kell változtatni, vagy optimalizálni a monitorozó eszközök adataira alapozva. Ugyanakkor, maga az üzleti folyamat viszont nem rendelkezik olyan eszközökkel, amelyek képesek együttműködni a monitorozó eszközökkel.

- Az üzleti szabályok és a döntési pontok a vállalaton belül több üzleti folyamatba is be vannak építve, így nem egyértelmű, hogy melyik folyamat melyik vállalati szabálytól is függ.

- Egy központi csoport készíti az vállalati szabályokat és irányelveket, és implementálás közben az egyes szabályokat minden folyamat résztvevői másként értelmezik, és egymásnak ellentmondóan valósítják meg őket.

- Az üzleti elemzőknek csak korlátozott rálátásuk van a fejlesztésre, és nem tudják módosítani a szabályokat, miután beépítették őket a folyamatba. Az IT szakemberek gyakran félreértelmezik a szabályokat.

- A vállalatnak nincs egy központi szabálygyűjteménye. Ezért nehéz az egyes irányelvek módosítása, illetve eme módosítások végig vitele a vállalat összes részlegén belül.

Az Oracle Business Rules lehetővé teszi, hogy az üzleti szabályokat az alkalmazás kódtól függetlenül adjuk meg, ezáltal az üzleti elemzőknek lehetőségük van arra, hogy gyorsan tudják változtatni a szabályokat grafikus eszközök segítségével.

Főbb részei:

- **Rule Author:** lehetővé teszi a felhasználói számára az üzleti alkalmazások létrehozását és módosítását egy form alapú felületen keresztül. Üzleti, nem pedig programozási

nyelvezetet használ. Egy általános célú alkalmazás, amely a Rule SDK-ra épül. Mind vékonykliensként, mind egy JDeveloper plug-in-ként támogatott.

- **Rules Language (RL):** egy teljes értékű programozási nyelv, amelyet a Java-val való integráció szem előtt tartásával fejlesztettek. Az RL interpretert Java-ban írták, és fel lehet használni Java programokon belül. Az RL-ből lehetőség van a szabályok létrehozására, módosítására és írására Java objektumokon keresztül, valamint Java függvényeket is meg lehet hívni. Inkább dekleratív mintsem procedurális, így könnyebb bizonyos típusú alkalmazások fejlesztése, valamint optimalizálási lehetőséget nyújt a rule engine számára.

- **Rule Engine:** a fő funkcionalitása magában foglalja a szabályok betöltését, a tényeknek munkaterületre történő bevonását, illetve az onnan történő eltávolításukat, valamint felületet nyújt a munkaterület megvizsgálására. Meghívható, mint egy Java könyvtár, vagy mint egy önálló szolgáltatás. Egy szabály tárházra támaszkodik, amely tartalmazza a szabályok halmazát, valamint egyéb meta információkat is. Ez lehet adatbázis vagy állományok.

- **Rule SDK:** támogatja a Java-ban írt, szabályokat módosító alkalmazások készítését, amelyek jellemzően vagy grafikus felülettel is el vannak látva, vagy böngészőkbe készülnek. Amiket nyújt:

    - egy könyvtári API, amely lehetővé teszi a fejlesztő számára, hogy üzleti objektumokat definiáljon, valamint szerkessze az üzleti felhasználók számára elérhető sablonokat.

    - egy API-t az üzleti folyamatok szerkesztésére, validálására és hibajavítására, illetve a korrekt futási idejű struktúrák generálására.

    - egy futási idejű API-t az üzleti szabályok betöltéséhez, végrehajtásához, monitorozásához és auditálásához.

Decision Service: egy módszer a szabályok, illetve szabályok halmazának publikálására, mint egy újrafelhasználható szolgáltatás, amelyet több üzleti folyamatból is meghívhatnak. Ez a szolgáltatás egy önálló egység, amely a következő részeket tartalmazza:

- Egy webszolgáltatás, amely becsomagolja a rules engine-t. Ez a szolgáltatás lehetővé teszi az üzleti folyamatok számára tények hozzáadását és eltávolítását. Egyes esetekben minden tényt, mint egy egységet kell hozzáadni, míg máskor inkrementális módon adják hozzá a tényeket. Ezért a szolgáltatásnak mind az állapot nélküli, mint az állapottal rendelkező interakciókat támogatnia kell.

- Szabályok, amelyeket ki kell értékelnie. Ezeket a Rule Author segítségével definiáljuk.

- Meta információk, amelyek leírják, hogy mely szabályok kiértékeléséhez mely tények megléte szükséges. Ezeket a tényeket XSD definíciókkal kell közzétenni.

- Adapterek, amelyek rendszeresen betöltik a tényeket a háttérrendszerekből (adatbázisokból, állományokból). Az adatokat vagy közvetlenül az üzleti folyamattól kapja, vagy az adapteren keresztül olvassa.

5. *OracleAS Integration Business Activity Monitoring*

Az Oracle BAM (Business Activity Monitoring) célja az döntéshozatali folyamat támogatása a vállalat pillanatnyi állapotának feltárásával. Segítségével valós időben követhetjük a vállalaton belül bekövetkező üzleti eseményeket, és ezt az információt felhasználva lehetővé válik az üzleti folyamatok hatékonyságának növelése.

Lehetőséget nyújt a vállalati vezetőknek arra, hogy nyomon kövesség az üzleti folyamatokat és a szolgáltatásokat a vállalaton belül, az egyes folyamatok KPI-it (Key Performance Indicator), és ami a legfontosabb, hogy lehetővé válik az üzleti folyamatok gyorsabb és hatékonyabb változtatása, így a megváltozott körülményekhez hatékonyabb tudnak alkalmazkodni, gyorsabban tudnak rá reagálni.

Amíg a beépített, alkalmazás szintű monitorozás alkalmas az események kiváltására, vagy jelzések generálására az egyes alkalmazások szintjén, addig általában nincs rá mód, hogy összehasonlítsuk a különböző alkalmazásokból származó eseményeket egymással, vagy épp az érintett üzleti folyamat eseményeivel. A BAM egy üzleti folyamat szintű monitorozási megközelítést alkalmaz, amely segíti az adminisztrátorokat, hogy a helyett, hogy a felmerülő következmény hibák sokaságával kellene megbirkózniuk, inkább a hibát kiváltó alap hibával

tudjanak foglalkozni, és arra tudjanak a lehető leghatékonyabban reagálni. Ha az adminisztrátorok számára nem lehetséges az okok és a következmények szétválogatása (amelyek együtt jelennek meg az eseményekben), akkor a valódi kiváltó ok megtalálása és kijavítása igen időigényessé válhat.

Az Oracle BAM egy üzenet alapú, esemény vezérelt, memória rezidens architektúrára épül, amelyet úgy terveztek, hogy kielégítse a valós idejű elemző és jelentő alkalmazások igényeit. Valós idejű betekintést nyújt a vállalat műveleteibe, és részletes elemzéseket nyújt az üzleti felhasználók számára, amelyek segítségével csökkenthetik a költségeket és javíthatják a folyamataikat, amikor maguk az események történnek. Egyesíti az értesítő, az adat integráló, adat cache-elő, elemző monitorozó, figyelmeztető és jelentő technológiákat, hogy a kívánt kritikus információkat másodpercekkel az esemény bekövetkezte vagy a státusz változása után ki tudja szállítani.

Négy fontos architekturális eleme van:

- **Adat- és eseménygyűjtő infrastruktúra:** különböző technikákkal segíti a felhasználókat, hogy elfogják a valós idejű eseményeket és információkat. Ez magában foglalja a szokásos és a dobozos alkalmazásokat, üzleti folyamatokat, adatbázisokat, üzenetkezelő rendszereket (JMS, AQ, stb.) és egyéb rendszereket. A BPEL Process Managerrel is teljesen integrált, hogy könnyebben tudja a folyamatokkal kapcsolatos adatokat kinyerni futás közben.

- **Esemény elemző és számító infrastruktúra:** segíti a felhasználókat, hogy szűrjék, összehasonlítsák és elemezzék az információt, hogy átláthatóvá váljon a hatásuk a definiált metrikákon.

- **Vizualizáció, intuitív műszerfal (dashboard):** egy web böngészőn keresztül egy interaktívan működő műszerfalat kapunk, amelyben valós idejű adatokat láthatunk. Segítségével gyorsan létre lehet hozni olyan jelentéseket, amelyekben nyomon követhetjük a kritikus üzleti értékek és KPI-k állapotát és változását. Riasztási feltételeket és szabályokat is lehet definiálni, amelyek értesítést küldenek, ha valamelyik bekövetkezett.

- **Valós idejű reagáló figyelmeztetések:** az Oracle BAM képes figyelmeztetéseket küldeni, ha az üzleti folyamatok állapota nem az elvártnak megfelelően alakul. A figyelmeztetések más eseményeket is kiválthatnak, beleértve a külső programok vagy webszolgáltatások meghívását is, hogy az üzleti folyamatot valós időben változtassa meg. A felhasználók a műszerfal segítségével is megtehetik a szükséges lépéseket.

6. Oracle Web Services Manager

Az Oracle WSM átfogó megoldást nyújt a szolgáltatásokkal kapcsolatos biztonsági problémák megoldására és kezelésére. Lehetővé teszi, hogy központilag definiáljunk irányelveket (policy) a webszolgáltatásokkal kapcsolatos műveletek irányítására, mint például a hozzáférések szabályozása (autentikáció és authorizáció), logolás és a tartalom validálása, és ezek után ezen irányelveket hozzá tudjuk kapcsolni a már létező webszolgáltatásokhoz úgy, hogy a szolgáltatásokhoz magukhoz nem kell hozzányúlnunk, változatlanul hagyhatjuk a megvalósításukat. Így lehetőség van úgy fejleszteni a szolgáltatásainkat, hogy magukban a konkrét megvalósításokban nem kell foglalkoznunk a jogosultságok kezelésével, nem kell ugyanazokat a részeket újra és újra implementálnunk, ezáltal is gyorsítva a fejlesztést.

Az Oracle WSM futási idejű adatokat is gyűjt, így lehetőség nyílik a hozzáférés szabályozás, üzenet integritás és a szolgáltatás minőségének (amelyet az SLA-kban [Service Level Agreement] definiálhatunk) monitorozására. Lehetőséget ad ezen értékek diagramokon történő megjelenítésére, amelyek segítségével az üzleti döntéshozók könnyebben dolgozhatnak, valamint ezek segítségével még időben közbe lehet lépni, ha a nyújtott szolgáltatás szintje nem éri el a vállaltat, ez által el lehet kerülni az SLA-k nem teljesítéséből fakadó többletkiadásokat az ügyfelek felé.

Összefoglalva, jobb irányíthatóságot és átláthatóságot nyújt a szolgáltatásaik állapotáról a vállalatok számára.

Kulcsfontosságú részei:

- **Policy manager:** egy grafikus eszköz az új biztonsági és műveleti szabályok megadására, tárolja a szabályokat és felügyeli a megosztásukat, valamint a szabály kikényszerítő pontok (átjárók [gateways] és ágensek [agents]) futásidejű frissítését. Lehetővé te-

szi az adminisztrátorok számára a műveleti szabályok konfigurálását és propagálását a megfelelő kikényszerítő eszközökig.

- **Enforcment:** a lehető legnagyobb rugalmasság érdekében az Oracle WSM két fajta szabály kikényszerítő komponenst definiál: átjárók és ágensek. Az átjárókat az alkalmazások és szolgáltatások egy csoportja előtt alkalmazzák, képesek megakadályozni az üzenetek átjutását annak érdekében, hogy kikényszerítsék a szabályok betartását, ezáltal adva egy biztonsági réteget a már telepített alkalmazások és szolgáltatások elé. Az ágensek képezik az „utolsó védelmi vonalat", ők közvetlenül egy szolgáltatáshoz vagy alkalmazáshoz kapcsoltan működnek.

- **Monitoring Dashboard:** információt gyűjt az átjáróktól és ágensektől a szabályok kikényszerítésével kapcsolatban, és az eredményt grafikus alakban jeleníti meg. Így az adminisztrátoroknak lehetőségük van alkalmazásonként minőségi követelmények beállítására (minimális követelmény, amely alá nem lenne szabad menni), és figyelmeztetést kapnak, amint egy alkalmazás megközelíti az előírt minőségi korlátot. Valós idejű betekintést nyújt az operátoroknak a webszolgáltatások teljesítményének, biztonságának, és a kritikus folyamatainak az állapotába. Ez által lehetőséget teremt az elvárt és az aktuális értékek vizsgálatára, valamint segíti az aktuális állapotnak megfelelő optimális döntések meghozását.

7. Oracle Application Server

Az Oracle Application Server támogatja a J2EE-t, a webszolgáltatásokat, valamint a web szabványokat. Tartalmaz egy portál rendszert, amellyel lehetővé válik az adatok megjelenítése, miközben maguk az adatok védve maradnak. Ezáltal feleslegessé válik a portálok készítése, így a fejlesztők más munkákon dolgozhatnak. Az Instant Portal egy kész alkalmazás, amelyhez csak az adatokat kell hozzáadni, és beépített keresési, valamint hírek (news) funkcióval is rendelkezik.

Segít leküzdeni az eltűnt, illetve elavult iratokkal kapcsolatos problémákat. Automatikus mentést lehet engedélyezni, így, amennyiben véletlenül töröltek egy fontos dokumentumot, akkor azt elő lehet keríteni a mentésből. Az elavult dokumentumokat könnyen lehet archiválni, és

nyomon lehet követni a változásokat, valamint definiálni lehet hozzáférési szabályokat, illetve követni lehet, hogy ki, mit és mikor változtatott meg.

Előkészített és tesztelt megoldást kínál a céges weboldalak létrehozásához. Modell-vezérelt eszközöket ad a web oldalak fejlesztéséhez, beépített elemekkel, elrendezésekkel. Lehetőség van a statikus és dinamikus oldalak elérésének gyorsítására a cache-ing engedélyezésével. Web-szerverként az Apache web szerverét használja.

V. BPEL példa

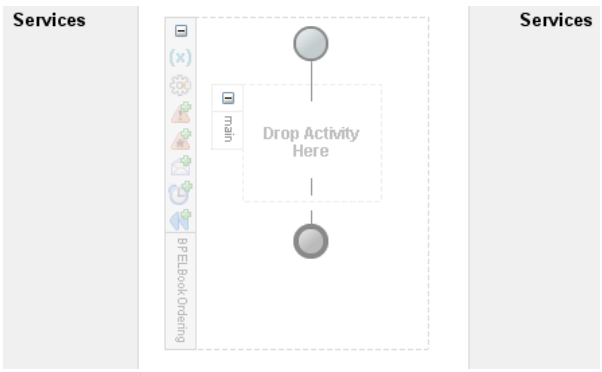
1. A feladat leírása

Legyen adott egy nagyvállalat, amely többek között használt könyvek árusításával is foglalkozik, és a könyveket antikváriumoktól szerzik be. Eme vállalatnak szüksége lehet egy olyan szolgáltatás nyújtására, amelyen keresztül tudnak az ügyfelek tőlük rendelni könyveket. A szolgáltatásnak a jobb árakat nyújtó antikváriumtól kell rendelnie. A feladat egyszerűsítése végett csak két antikváriummal állnak kapcsolatban, amelyek közül az egyik szinkron, a másik aszinkron szolgáltatáson keresztül nyújtja a könyvek árait. Feltételezzük, hogy a vevő mindenképp meg akarja venni a könyveket, illetve feltételezzük, hogy a megadott adatok alapján elvégezhető a rendelés (például eltekintünk a rendelkezésére álló pénzösszegtől, amely egy valós alkalmazás esetén nem tehető meg minden esetben, megtehető viszont akkor, ha utánvéttel veszi az árut).

2. A feladat megoldása JDeveloper segítségével

A *File* menüben válasszuk a *New…* menüpontot. A *Filter By:*-nál válasszuk az *All Technologies*-t, majd a *Categories*-en belül válasszuk a *General → Projects → BPEL Process Project*-et. A megjelenő ablakon válasszuk ki az *Empty BPEL Process*-t.

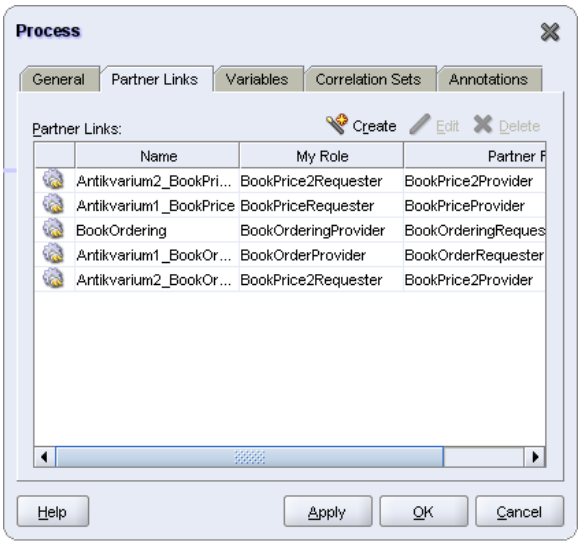
Most van egy üres BPEL folyamatunk, amely egy *main* nevű szekvenciából áll.



3. ábra Az üres szekvencia

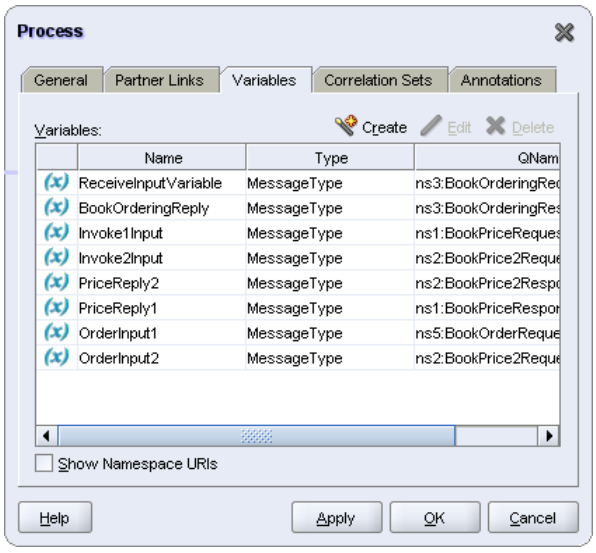
A process-en duplán kattintva a *Partner Links* fülön a *Create* varázsló segítségével hozzáadhatjuk a használni kívánt szolgáltatásokat. Nekünk jelenleg 5 szolgáltatásra van szükségünk: a

2-2 antikvárium által adottra (ajánlat kérés, illetve rendelés), illetve az általunk nyújtottra. Mindegyikhez szükséges a megfelelő WSDL állomány megléte.



4. ábra Az általunk ismert szolgáltatások

A folyamat során nyolc változót fogunk használni, ezek: *ReceiveInputVariable* (a szolgáltatásunkat meghívó által adott értékek), *BookOrderingReply* (az általunk a kliens felé adott válasz). Az első és a második antikvárium árajánló szolgáltatásának paraméterei: *Invoke1Input*, *Invoke2Input*. Az első és a második szolgáltatáshívás által nyújtott válasz: *PriceReply1*, *PriceReply2*. Az első és a második antikvárium által nyújtott rendelési szolgáltatás paraméterei: *OrderInput1* és *OrderInput2*.

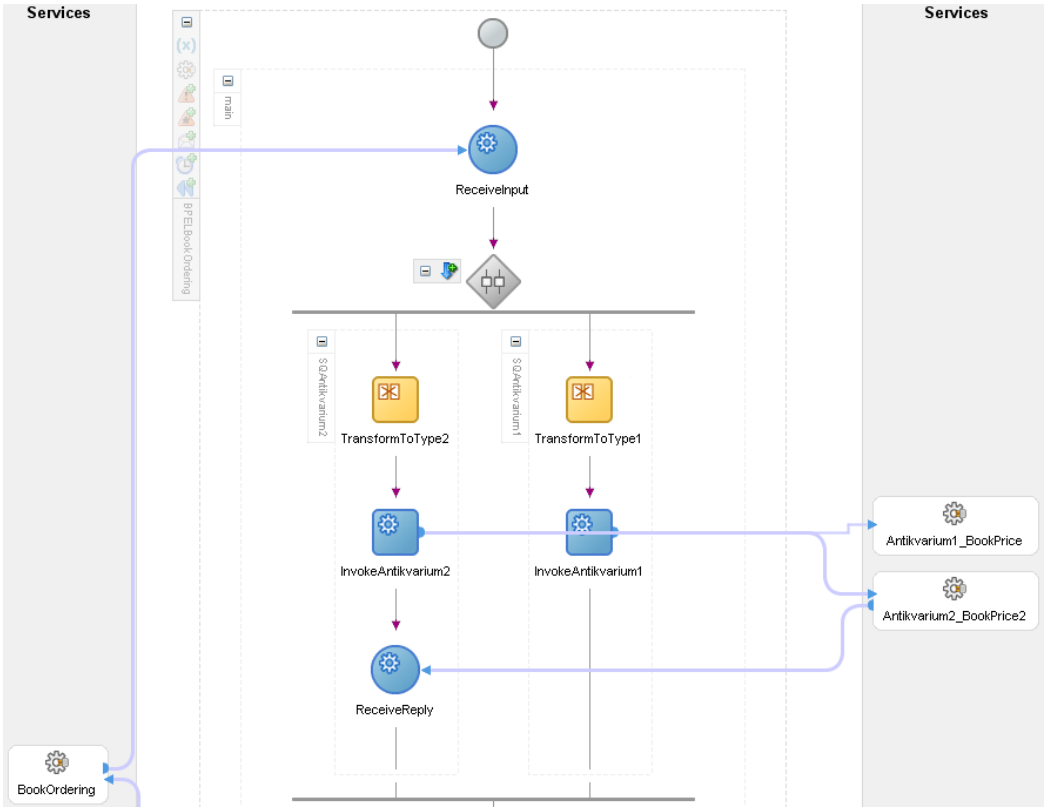


5. ábra A használt változók

A *Process Activities*-ben található elemekből összeállítjuk a folyamatot (a 6-os és 7-es ábra mutatja a teljes folyamatot).

A folyamat a *ReceiveInput* művelettel kezdődik, ami során a klienstől (*BookOrdering*) fogadjuk a kérést, és a kérésben található értékeket a *ReceiveInputVariable*-be másoljuk. Ez után párhuzamosan meghívjuk mindkét antikvárium szolgáltatását, és a szolgáltatáshívások előtt a megfelelő alakra hozzuk a kapott értékeket. A hívásokat a *Flow* műveletbe rakjuk, mivel a két hívás elvégezhető párhuzamosan is. Mivel az Antikvárium1 által nyújtott szolgáltatás a jelen példánkban szinkron, az Antikvárium2 által nyújtott pedig aszinkron, és a két hívás nem függ egymástól, ezért ajánlatos őket párhuzamosítani, így nem kell egymásra várniuk a kezdéssel.

Mivel az egyes szolgáltatások eltérő adatformátumot várnak, ezért szükséges a változók *ReceiveInputVariable*-el történő transzformációja a megfelelő alakra az egyes hívások előtt.

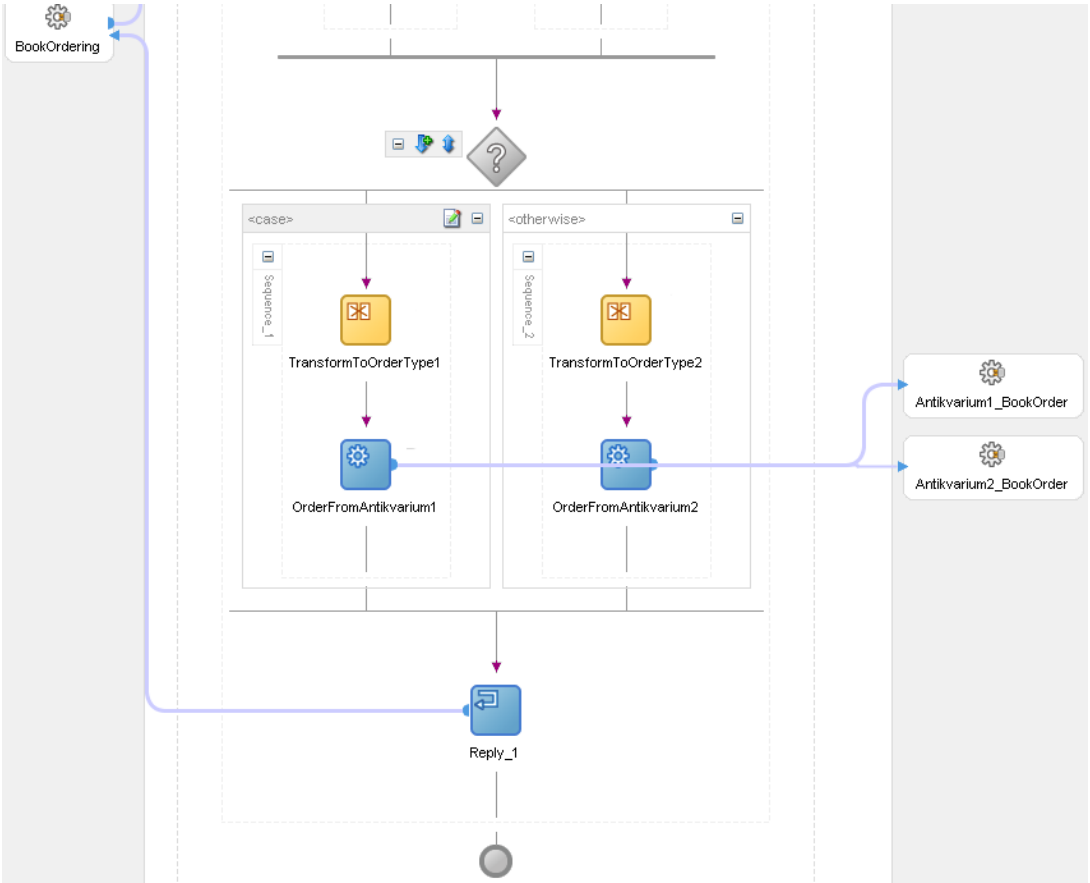


6. ábra A folyamat első fele

Miután megkaptuk mindkét szolgáltatástól a választ, egy *Switch* művelet segítségével eldöntjük, hogy melyik antikváriumtól fogunk rendelni, amelyhez a következő XPath kifejezést használjuk:

```
bpws:getVariableData('PriceReply1', 'price') < bpws:getVariableData('PriceReply2',
'price')
```

A *getVariableData* a BPEL névtérben definiált függvény, amely jelen esetben a megadott változóból kiveszi a megadott adattagjának az értékét. A mi feltételünk az, hogy a kisebb árat ajánlót fogjuk választani. Amennyiben a feltétel teljesül, akkor meghívjuk az *OrderFromAntikvarium1* műveletet, különben az *OrderFromAntikvarium2*-t. Miután ez befejeződött, értesítjük a klienst arról, hogy végeztünk (feltételezzük, hogy az antikváriumok megfelelő szolgáltatásai elvégzik a klienssel való kommunikációt).



7. ábra A folyamat második fele

VI. Összefoglalás

A webszolgáltatások napjaink üzleti informatikai világának egyik kulcsfontosságú részévé vált. Köszönheti mindezt annak, hogy széles körben támogatott, szabványos eszközökre épül, XML alapú kommunikációt valósít meg (ezáltal az egyes rendszerek közötti különbözőségeket is elfedi), és ami a legfontosabb, hogy üzleti szintű komponensek, szolgáltatások fejlesztését és elérését teszi lehetővé.

Az olyan eszközöknek köszönhetően, mint az ESB és a BPEL, lehetővé vált a szolgáltatások szintjén történő hatékony, akár az üzleti elemzők által, a programozók kihagyásával végzett üzleti folyamatok fejlesztése. Mivel ezek az eszközök lehetővé teszik az üzleti elemzők számára a folyamatok saját kezű módosítását, ezért gyorsabban tudnak reagálni az üzleti életben bekövetkező változásokra, ez által pedig anyagi haszonra tehetnek szert. Mivel ezek az eszközök könnyen kezelhetőek, nem szükséges hozzájuk komoly informatikai előképzettség, ezért az üzleti folyamatok és szolgáltatások szintjén a felső vezetés is bepillantást nyerhet eme folyamatok megvalósításába, illetve működésük mikéntjébe.

Az Oracle vállalat által nyújtott adatbázis kezelő rendszer megfelelő hátteret biztosít a vállalatok számára a nagy mennyiségű adatok tárolására, valamint az általuk készített SOA Suite részei támogatják a szolgáltatás-orientált programozást mind a fejlesztők, mind az üzleti elemzők számára.

Ezen szoftvercsomag egyes elemeinél szem előtt tartották azt a célt, hogy egymással a lehető leghatékonyabb módon tudjanak együttműködni, a lehető legjobb teljesítményt tudják nyújtani, és így támogassák az üzletek igényeit.

Az Oracle ESB egy teljes értékű Service Bus, segítségével összekapcsolhatjuk az üzleti folyamatainkat és alkalmazásainkat.

Az Oracle BPEL Process Manager segítségével modellezhetjük üzleti folyamatainkat, dekleratív módon fejleszthetjük őket.

Az Oracle Business Rules segítségével szabályozhatjuk az általunk nyújtott szolgáltatások elérhetőségét, biztonsági protokollokat definiálhatunk.

Az Oracle Web Service Manager egy kompakt eszközt nyújt a vállalatunkban lévő folyamatok pillanatnyi állapotába történő betekintéshez, így proaktív módon megelőzhetjük a hibák kialakulását, valamint értesülhetünk róla, ha az egyes szolgáltatásaink szintje az általunk az SLA-ban elvállalt érték alá kerül, így módosításokat eszközölhetünk ennek változtatására, amely segítségével fontos anyagi kiadásoktól mentesülhetnek a vállalatok.

Bízom benne, hogy eme dolgozat keretein belül sikerült rávilágítanom a webszolgáltatások hasznosságára, valamint, hogy sikerült tisztáznom az egyes Oracle eszközökkel kapcsolatos alap információkat.

Hivatkozások

[1.] Web Service, Wikipedia, http://en.wikipedia.org/wiki/Web_service

[2.] W3C, WSDL specifikáció, http://www.w3.org/TR/wsdl

[3.] W3C, SOAP specifikáció, http://www.w3.org/TR/soap/

[4.] Oracle, Oracle SOA Suite Quick Start Guide, http://download.oracle.com/docs/cd/B31017_01/core.1013/b28938.pdf

[5.] Oracle, Oracle SOA Suite Developer's Guide, http://download.oracle.com/docs/cd/B31017_01/core.1013/b28764.pdf

[6.] Oracle, Oracle BPL, http://www.oracle.com/technology/products/ias/bpel/htdocs/pdf/orabpel-bpel101.pdf

[7.] Oracle, Oracle Application Server 10g, http://www.oracle.com/technology/products/ias/pdf/as_se1_datasheet.pdf

[8.] Charlton Barreto, BPEL, http://charltonb.typepad.com/weblog/2003/08/what_is_bpel4ws.html