



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Halász László

**E-BUSINESS FOLYAMATOK
MEGVALÓSÍTÁSA IBM
WEBSHERE TERMÉKEKKEL**

KONZULENSEK

Géczy Viktor

Huszerl Gábor

BUDAPEST, 2009

HALLGATÓI NYILATKOZAT

Alulírott **Halász László**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Tudomásul veszem, hogy az elkészült diplomatervben található eredményeket a Budapesti Műszaki és Gazdaságtudományi Egyetem, a feladatot kiíró egyetemi intézmény saját céljaira felhasználhatja.

Kelt: Budapest, 2009. 05. 15.

.....
Halász László

Tartalomjegyzék

Összefoglaló	6
Abstract.....	7
1 Elektronikus kereskedelmi rendszerek megoldásai.....	8
1.1 Integráció	8
1.2 Egyéb szabványok	20
1.3 Kapcsolódó IBM termékek	22
2 Rendszertervezés.....	33
2.1 A rendszer funkcionális leírása	33
2.2 A rendszer működésének alapvető koncepciói	34
2.3 A rendszerben előforduló forgatókönyvek	40
2.4 Rendszer komponensei és kapcsolata	43
3 Megvalósítás	47
3.1 Adattáblák bemutatása	47
3.2 MQ infrastruktúra bemutatása	48
3.3 ESB beállítások és folyamatok bemutatása	50
3.4 A kliensoldali kód ismertetése	61
3.5 Továbbfejlesztési lehetőségek	67
4 Összefoglalás.....	68
Irodalomjegyzék.....	69
Függelék.....	70

Összefoglaló

A diplomaterv célja az elektronikus kereskedelmi rendszerekben használatos technológiák áttekintése és egy példa alkalmazás megtervezése, megvalósítása. Munkám során tanulmányoztam az Enterprise Service Bus, Message Queue, Web Services technológiákat, megismerkedtem az IBM WebSphere termékcsalád egy részével, és az így szerzett tudást alkalmaztam egy minta elektronikus kereskedelmi rendszer megvalósításában.

A diplomaterv első fejezete sorra veszi az elektronikus kereskedelmi rendszerekben használatos technológiákat, tervezési mintákat, a második fejezete tartalmazza a rendszer funkcionális leírását és bemutatja a rendszer kialakítása közben felmerülő megvalósítási lehetőségeket és a meghozott tervezői döntéseket. Az implementáció egyes lépéseit dokumentáltam a harmadik fejezetben, a függelék pedig a kapcsolódó konfigurációs és forráskód bejegyzéseket tartalmazza.

Abstract

The main goal of this document was to give a proper revision of the technologies used in the different types of e-bussines systems and the design and implementation of an application. During my work I got to know the Enterprise Service Bus, Message Queue and Web Services technologies and I also got familiar with a special part of the IBM WebSphere product family. I applied this knowledge at the implementation of a sample electronic commerce system.

The first part of the document deals with the most widespread technologies and design patterns used in e-commerce systems. The second part consists of the functional description of the system and also shows the possible implementation opportunities and the decisions made during the design of the system.

The steps of the implementation process are documented in the third part of the document while the connected configuration settings and source code can be found in the appendix.

1 Elektronikus kereskedelmi rendszerek megoldásai

A fejezet célja az elektronikus rendszerek kialakítása során gyakori feladatként felmerülő integráció bemutatása, az integrációs minták ismertetése és a hozzá kapcsolódó korszerű szabványok (Enterprise Service Bus , Web Services) bemutatása. A fejezet tartalmazza a diplomaterv feladatához használt kliensoldali technológiák (JavaScript, AJAX), valamint IBM termékek (IBM WebSphere Enterprise Service Bus, IBM WebSphere Message Queue) rövid bemutatását.

1.1 Integráció

Nagyvállalati környezetben az informatikai infrastruktúrával szemben támasztott egyik fontos elvárás, hogy a működéshez szükséges valamennyi funkciót biztosítsa, valamint a környezeti hatásokra kellőképpen gyorsan és költséghatékonyan reagáljon. Általában nem igaz, hogy egyetlen alkalmazás biztosítani tudná az összes szükséges funkciót, mint ahogy az sem, hogy az alkalmazások izoláltan hatékonyan tudnának működni. Az alkalmazások funkcióinak hiányosságainak kompenzálására adódik az integráció ötlete. Integráció segítségével létrehozhatóak olyan rendszerek, melyek a meglévők képességeit felhasználva komplexebb funkciókat biztosítanak a felhasználók számára, növelve ezzel a munkavégzés hatékonyságát. Az integrációt számos körülmény nehezítheti.

Ilyenek lehetnek az :

- eltérő platformok és kommunikációs protokollok
- eltérő interfészek
- inkompatibilis szabványverziók
- fájlformátum változások
- különböző rendelkezéseállási jellemzők
- akár fejlesztési időben is változó igények

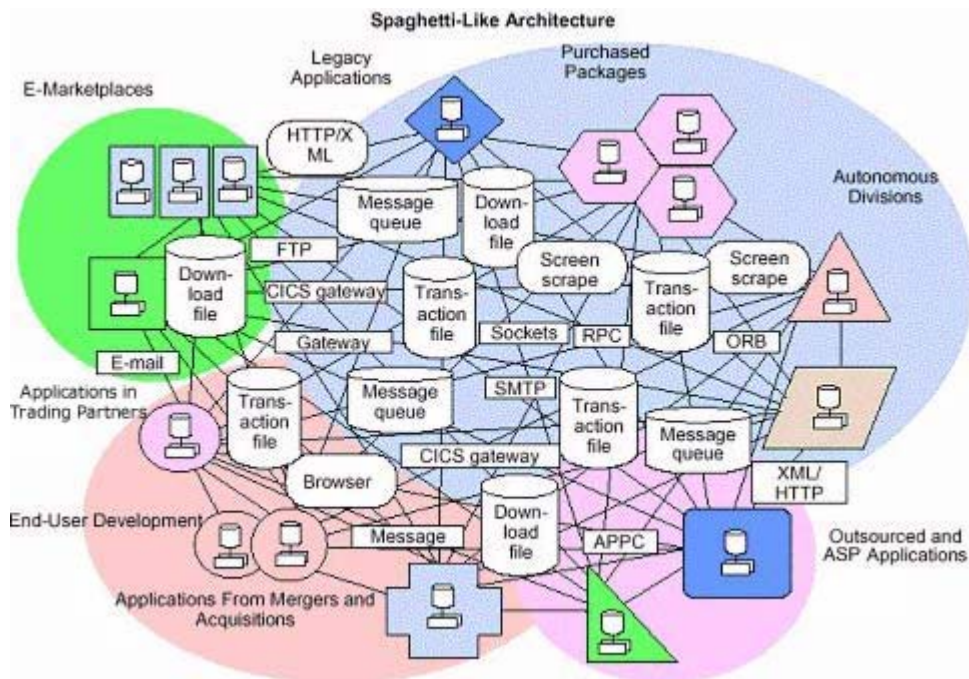
Ha ráadásul az integráció hatóköre túlmutat egy vállalat keretein, akkor a probléma csak tovább bonyolódik.

1.1.1 Integrációs minták

A rendszerek és alkalmazások integrációjának számos módja létezik. A fejezet célja ezen minták bemutatása.

Közvetlen kapcsolat

Kezdetben az alkalmazások között pont-pont kapcsolatokat alakítottak ki, azaz minden alkalmazás közvetlenül kommunikált a másikkal. Ezek a program hidak az aktuális feladat függvényében fájlkon, alkalmazás programozói interfészekon (API) vagy valami egyéb egyedi bináris megoldáson alapultak. Kifejlesztésük munkaigényes volt, az alkalmazott program híd ritkán volt más feladtnál is alkalmazható, a komponensek változásai esetén jó eséllyel újra le kellett gyártani az új implementációnak megfelelően, és sok komponens együttműködése esetén a rendszer fejlesztését már a nehéz áttekinthetőség is bonyolította. Ennek az integrációs minta alkalmazásának az eredménye az 1. ábrán látható „spagetti architektúra”.



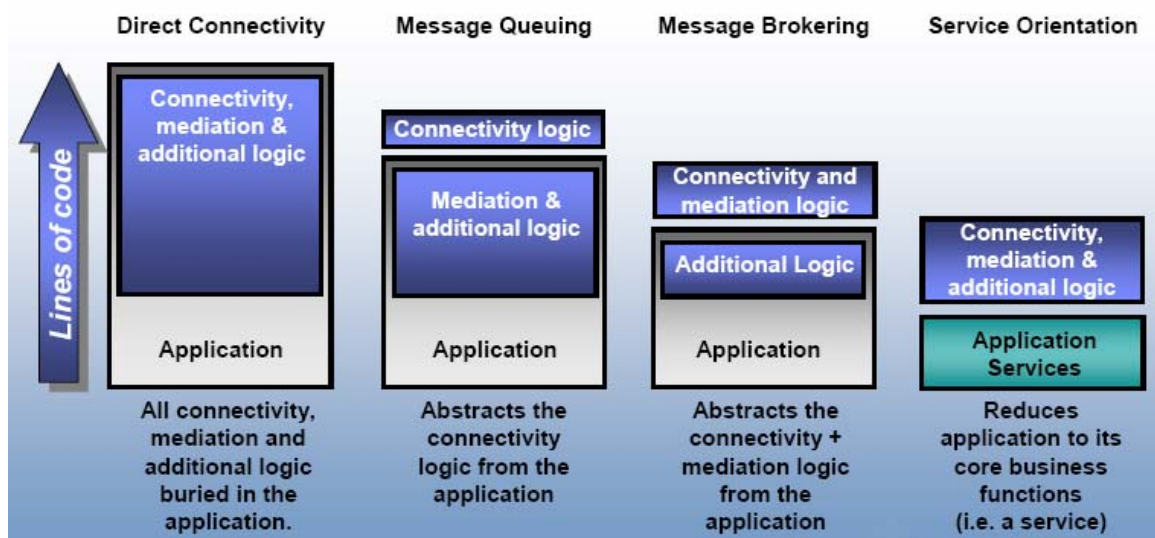
1. ábra Spagetti architektúra [1]

A probléma felismerése vezetett az architektúrális fejlődéshez. A hatékonyabb fejlesztés érdekében olyan komponensekből akartak a fejlesztők építkezni, melyeknél a komponensek szabványos, platformfüggetlen módon kommunikálnak, az egyes feladatokat független modulok végzik el, így a létező szolgáltatások

újrhasználhatóak, a változások gyorsan implementálhatóak modulok cseréjével, s az egész infrastruktúra jobban menedzselhetővé válik.

Az alkalmazásintegráció három típusú logika megvalósításával jár, ezek a kapcsolódási, a közvetítői és az egyéb járulékos logika. A kapcsolódási logika többek között azt definiálja, hogy hogyan éri el egymást a kommunikáló alkalmazások, milyen protokollt használnak az átvitelre, és hogy hogyan kezeljék a másik fél esetleges elérhetetlenségét. A közvetítői logikába a kapott adat feldolgozása, átalakítása, esetleges továbbküldése tartozik. A járulékos logika azon műveleteket és beállításokat tartalmazza, mely az előző két kategóriába nem tartozik bele, de az integráció során felmerül.

Közvetlen kapcsolat esetén az alkalmazásba bele van kódolva mind a kapcsolódási, a közvetítői, mind pedig az egyéb járulékos logika, nehezítve ezzel ennek újrhasználhatóságát. A végbemenő architektúráis fejlődést a 2. ábra mutatja. Az ábrán látható, ahogy az egyes feladatok egyre inkább elkülönülnek az alkalmazástól lecsökkentve ezzel a fejlesztő által megírandó kód hosszát, így felgyorsítva a fejlesztést.

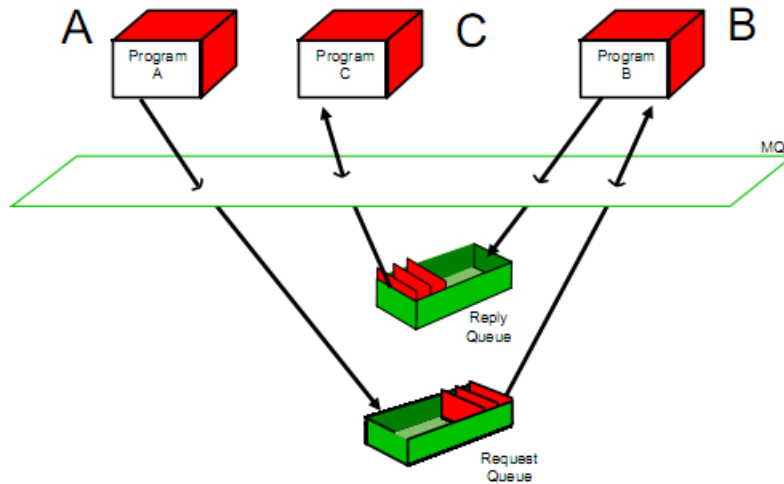


2. ábra Az integrációs architektúra fejlődése [2]

Üzenetsorok használata

Üzenetsorok használatakor az alkalmazások lazán csatoltan, üzenetek segítségével kommunikálnak. Az interakció folyamán az üzenetet egy külső alkalmazás által biztosított várakozási sorba helyezi a küldő, a fogadó alkalmazás pedig leveszi onnan. Ezzel a kapcsolódási logika elkülönül az alkalmazástól. A 3. ábrán látható, ahogy az A, B és C program az MQ interfészen (MQI) keresztül eléri a "Request" és

“Reply” üzenetsorokat, és üzenetek elhelyezésével aszinkron módon kommunikálnak egymással. Az MQ interfésznek számos nyelven létezik megvalósítása, így az egymással üzenetsorok segítségével kommunikáló alkalmazások akár különböző nyelven megvalósíthatóak.



3. ábra Üzenetsorok használata [3]

Az üzenetsorokat biztosító alkalmazás plusz szolgáltatásként biztosíthatja a kommunikáció során felmerülő tipikus problémák megoldását.

Ide sorolhatóak az alábbiak:

- megbízható üzenetküldés, azaz garantálhatja, hogy minden küldött üzenet megérkezik a címzethez, akár annak átmeneti kiesése esetén is,
- az üzenetek sorrendhelyes vétele, azaz hogy az üzeneteket abban a sorrendben kapja meg a fogadó fél, ahogy a küldő küldte őket,
- a duplikált üzenetek szűrése, azaz hogy a kommunikációs csatorna hibájából duplikálódott üzenetekből is csak egyet kapjon meg a fogadó fél.

Ez a fejlesztőket sok munkától kíméli meg, viszont a közvetítői logika továbbra is az alkalmazás része.

Üzenet közvetítés

Üzenet közvetítésnél a kapcsolódási és a közvetítői logika is az alkalmazástól különböző rétegben helyezkedik el, az alkalmazásban már csak az egyéb járulékos logika kódja található. Az üzenetközvetítő alkalmazás támogatja az üzenet tartalmának módosítását valamint az üzenetnek a tartalma alapján történő irányítását.

Szolgáltatás-orientált architektúra

A szolgáltatás orientált architektúrában (továbbiakban SOA) az alkalmazásban már tisztán csak az üzleti logika található, a járulékos kapcsolódási és közvetítői logika már tőle elkülönül.

A szolgáltatás orientált architektúra az informatikában különböző szempontok szerint mást és mást jelent.

- Megvalósítás szempontjából egy fejlesztési modell, mely szabványokra épül, megfelelő módszertani- és eszköztámogatással bír, s korszerű technológiákat alkalmaz.
- Architekturális szempontból egy olyan stílus, melynek jellemzői a laza csatolás, az újrafelhasználás és az egyszerű és összetett szolgáltatások kombinációja. Három fő eleme a szolgáltató (service provider), az igénybevevő (requestor) és a leíró (descriptor).
- Üzleti szempontból azt jelenti, hogy az üzletet illetve ügymenetet szolgáltatások halmazaként tudjuk leképezni, s könnyen elérhetővé tesszük ügyfeleink és partnereink számára.
- Az üzemeltetés szempontjából szolgáltatási szintű megállapodásokat (SLA) jelent, melyek biztosítják a szolgáltató és az igénybevevő közötti, üzleti prioritásoknak megfelelő szolgáltatás-minőség (QoS) betartását.

A SOA tehát egy építkezési paradigma elosztott rendszerekhez. Az alkalmazások funkcionálisai szolgáltatásokként érhetőek el, így a végfelhasználói alkalmazások fel tudják használni azokat illetve más komponensek is építhetnek az így elérhető adatokra. A SOA-ban minden alkalmazás ugyanazon a platformfüggetlen módon kommunikál a másikkal, így bármely alkalmazás, mely el tudja érni a másik – például web szolgáltatásként – kiajánlott szolgáltatását, fel tudja használni azt függetlenül annak konkrét implementációjától.

A szolgáltatás-orientált architektúrát érdemes alkalmazni, ha rendszerünkben az alábbi szempontok fontosak:

- helyátlátszóság: a szolgáltatások nem kötődnek konkrét rendszerhez vagy hálózathoz
- jobb kódúrafelhasználás
- gyorsabban összeállítható egy új funkcionalitás

- biztonsági szolgáltatások laza csatolása, így a meglévő alkalmazáshoz autentikáció és autorizáció könnyen illeszthető
- a szolgáltatások dinamikus kereshetősége és hívhatósága

A SOA számos előnye ellenére nem minden esetben a legjobb választás. Ilyen esetek például:

- stabil, homogén vállalati rendszerben nem költséghatékony a bevezetése
- ha nem kínálunk fel kifelé, illetve nem veszünk igénybe külső szolgáltatásokat
- real-time jellegű követelmények esetén
- időkritikus rendszerek esetén

1.1.2 Web szolgáltatások

A web szolgáltatások olyan szoftver komponensek, melyek szolgáltatásokat biztosítanak, interfészüket saját web szolgáltatás leíró nyelvük (WSDL – Web Services Description Language) írja le, és olyan standard hálózati protokollokon keresztül érhetőek el, mint például a SOAP HTTP felett. A web szolgáltatások technológiát olyan nagy cégek fejlesztették ki mint az IBM vagy a Microsoft, és a W3C valamint az OASIS szabványosító szervezetek szabványosították. Ez a technológia ideális platform- és protokollfüggetlen eszköze a különböző szervezetek közötti együttműködésnek, integrációnak.

XML

Egy web szolgáltatással XML alapú üzenetek segítségével lehet kommunikálni. Az XML (Extensible Markup Language, Kiterjeszhető Leíró Nyelv) a W3C által ajánlott általános célú szabvány, speciális célú leíró nyelvek létrehozására. Egy XML állomány elemekből áll, melyek hierarchikus szerkezetben, fa struktúrában helyezkednek el. Egy elem tartalmazhat:

- más elemeket
- szöveges tartalmat
- mindkettőt
- semmit

Minden elem rendelkezhet attribútumokkal. Az attribútumok általában az elem szöveges tartalmához tartozó metaadatokat tartalmazzák. Az elemek csoportosítására névterek megadásával van lehetőség.

[4]Az XML azon tulajdonságai, amelyek alkalmassá teszik adattovábbításra:

- mind ember, mind gép számára olvasható formátum
- támogatja a Unicode-ot, ami lehetővé teszi bármely információ bármely emberi nyelven történő közlését
- képes a legtöbb általános számítástudományi adatstruktúra ábrázolására (rekord, lista, fa...)
- öndokumentáló formátum, amely struktúra- és mezőneveket ír le speciális értékekkel együtt
- szigorú szintaktikus és elemzési követelményeket támaszt, ami biztosítja, hogy a szükséges elemzési algoritmus egyszerű, hatékony és ellentmondásmentes maradjon

Az XML-t gyakran használják dokumentumtárolási és feldolgozási formátumként, mind online mind offline módon, és több előnnyel is jár:

- internetes szabványokon alapuló erőteljes, logikailag ellenőrizhető formátum
- a hierarchikus struktúrája megfelel a legtöbb (de nem mindegyik) dokumentum típusnak
- egyszerű szöveg formátumban valósul meg, licencektől és korlátozásoktól mentesen
- platform-független, így viszonylag immunis a technológiai változásokkal szemben
- az XML-t és elődjét, az SGML-t már több mint tíz éve használják, így széles tapasztalat és eszközkészlet áll rendelkezésre

Egy XML állományt két szempontból értékelhetünk. Ezek a szempontok a helyesség és az érvényesség. Egy XML állomány helyes, ha megfelel az XML szintaxis szabályainak, és érvényes, ha ezen felül még egy sémaleírónak is megfelel. Egy sémaleíró azt írja le, hogy az XML állomány struktúrájára milyen megkötések vannak. Séma definiálására alkalmas szabvány például a következő fejezetben említett XSD.

XSD

Az XSD (XML Schema Definition – XML Séma Definíció) XML állomány. Segítségével XML állományok struktúrájára és tartalmára fogalmazhatóak meg

követelmények. Amennyiben az állomány megfelel a definiált követelményeknek, akkor érvényes. Az XSD-vel a következő típusú tulajdonságok rögzíthetőek:

- megadható az elemek típusa előre definiált típusok közül
- elemek értékkészlete
- elemek és attribútumok kezdeti és fix értéke
- elemek és attribútumok minimális és maximális értéke
- elemek és attribútumok minimális és maximális hossza
- attribútumok kötelezővé tétele
- összetett elemek részlemeinek megadása
- összetett elemek részlemeinek sorrendje, gyakorisága

XSD-k hivatkozhatnak egymásra, így összetett sémadefiníció is kialakítható velük.

WSDL

A WSDL egy XML alapú leíró, mely tartalmazza egy web szolgáltatás meghívásához szükséges paramétereket. Ezek közé tartozik hogy az adott szolgáltatás milyen műveleteket kínál fel, a műveletekhez milyen adattípusok kapcsolódnak és hogy milyen címen és protokollon keresztül hívhatóak meg. A WSDL-t általában program generálja, de természetesen a kézi szerkesztésére is van lehetőség.

SOAP

Web szolgáltatás meghívása közben az adatok SOAP üzenetek formájában utaznak a hálózaton. Egy érvényes SOAP üzenet egy jól formázott XML állomány. Gyökéreleme a SOAP envelope nevű elem, mely opcionálisan tartalmaz egy SOAP Header és kötelezően egy SOAP Body elemet. A SOAP Header tipikusan metaadatokat tartalmaz, mely például titkosítás esetén a választott algoritmust tartalmazza. Az üzenet további része – például a web szolgáltatás hívásának paraméterei – a SOAP Body részben helyezkednek el.

Egy példán szemléltetve, az alábbi szignatúrájú függvény meghívása:

```
int doubleAnInteger ( int numberToDouble );
```

a 123-as bemenő paraméterrel az alábbi kérésüzenetet generálja:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```

xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnInteger
      xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">123</param1>
    </ns1:doubleAnInteger>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Az üzenet láthatóan nem tartalmaz fejléct, mind a meghívandó függvény neve mind pedig a paramétere az üzenet törzsében található. A meghívandó függvény neve az ns1 prefix-szel a doubleAnInteger elem reprezentálja, míg hívási paraméterét a param1 nevű elem tartalmazza.

UDDI

A web szolgáltatások regisztrációs támogatás nélkül olyan nehezen használhatóak lennének, mint az Internet keresők nélkül, ezért definiálták a szolgáltatások regisztrációjára alkalmas szabványt, a UDDI-t (Universal Description, Discovery, Integration). Egy UDDI tárbn a szolgáltatók az általuk biztosított szolgáltatásokat regisztrálhatják, valamint a szolgáltatást igénylők megkereshetik a számukra megfelelő szolgáltatást. A fenti folyamat látható a 4. ábrán. A szolgáltatás nyújtó (Service Provider) az első lépésben regisztrálja szolgáltatását a UDDI tárbn (Register Service Contents), melyet a szolgáltatás igénylője (az ábrán a User) megkeres (Search Service), majd miután megtalálta felhasznál (Use Service).



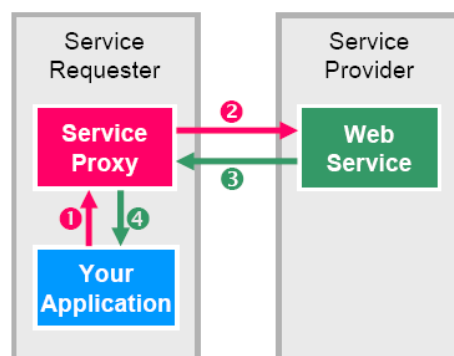
4. ábra Kapcsolatfelvétel UDDI segítségével [5]

Egy UDDI szolgáltatás bejegyzés például a szolgáltatás nevét, leírását, jellegét, a szolgáltatáshoz tartozó WSDL fájl URL címét és egyéb kapcsolódó információkat

(például QoS paramétereket) tartalmazhat. Lehet nyilvános vagy privát egy nagyvállalat számára. Hátránya, hogy hatékonysága erősen függ karbantartottságától, így többek között attól, hogy a megszűnt szolgáltatásokat eltávolítják-e belőle.

Szolgáltatás proxy

Egy web szolgáltatás meghívása az úgynevezett szolgáltatás proxy-n keresztül történik. A szolgáltatás proxy-t a szolgáltatás igénylője generálja a WSDL alapján futási vagy fejlesztési időben. A proxy a szolgáltatás igénylőnél fut. Amikor a szolgáltatás igénylője meghívja a szolgáltatást, akkor gyakorlatilag a szolgáltatás proxy egy metódusát hívja meg. A proxy készíti el a SOAP üzenetet a kérés alapján, mely tartalmazza a hívott szolgáltatás nevét és argumentumait, és elküldi ezt a szolgáltatás nyújtójának. Az kiértékeli az üzenetet, végrehajtja a kérést, majd – amennyiben kétirányú metódust hívott meg a szolgáltatás igénylő – elkészíti a választ és visszaküldi. Ezután a proxy a kapott üzenetet visszaalakítja a szolgáltatás igénylő natív kódjára, és a kliensnek átadja. Így működhet a különböző platformú szolgáltatások és kliensek hatékony kommunikációja, mely nyomonkövethető az 5. ábrán.



5. ábra Web szolgáltatás meghívása [6]

Amennyiben a szolgáltatás extra funkciókat biztosít, például megbízható vagy titkosított üzenetküldést, akkor az ehhez szükséges műveleteket a proxy transzparens módon elvégzi, komoly munkát levéve ezzel a fejlesztő vállaról.

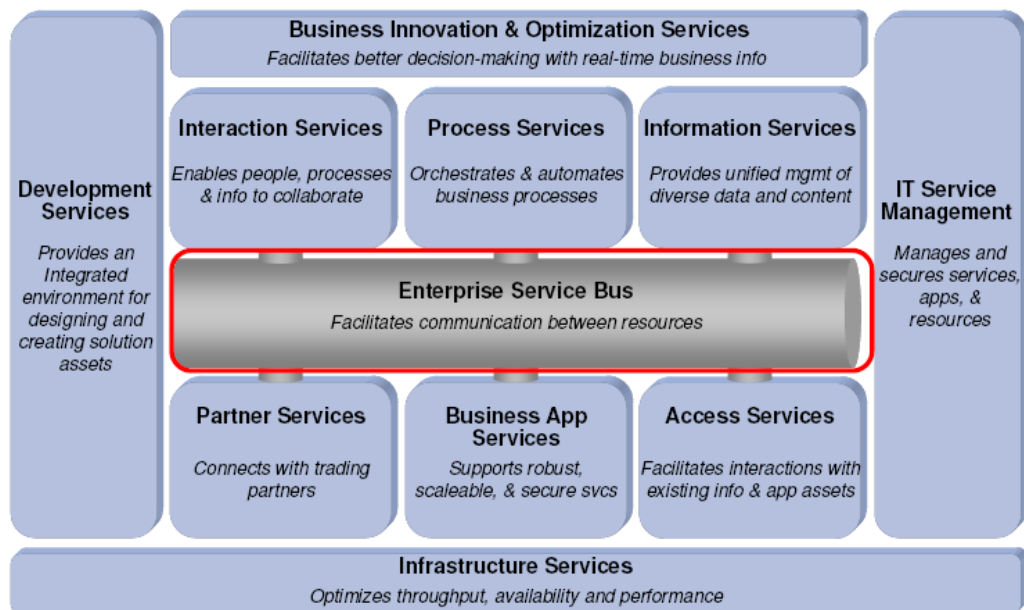
1.1.3 Enterprise Service Bus

Az Enterprise Service Bus egy architektúrális minta. Szabványokon alapuló integrációt tesz lehetővé lazán csatolt alkalmazások között, így szolgáltatás-orientált architektúrában, üzenet-vezérelt architektúrában és eseményvezérelt architektúrában egyaránt. Az ESB-n belül található kiemelt fontosságú komponens a mediációs

komponens. Feladata az üzenetek, események, szolgáltatáshívások intelligens feldolgozása. A mediációs komponens a busz infrastruktúrában az alábbi kulcsképessegekkel rendelkezik:

- tartalom alapján irányítja az üzeneteket
- transzformálja az üzeneteket

Az Enterprise Service Bus helyét a szolgáltatás orientált architektúrában és a modulokkal való kapcsolatát a 6. ábra mutatja. Láthatóan a feladata az erőforrások közötti kommunikáció biztosítása a fő feladata.



6. ábra Szolgáltatás orientált architektúra [7]

Az Enterprise Service Bus egyfajta intelligens lazán csatolt kapcsolatot teremt a szolgáltatást igénylők és a szolgáltatást biztosítók között. Ennél a kapcsolatnál szolgáltatás virtualizációról beszélhetünk, ugyanis az Enterprise Service Bus elfedi a hívó elől a szolgáltatót, konverziós képességének köszönhetően a hívás módja szintén rejtve marad, valamint transzformációk segítségével a szolgáltatások interfészeitől is függetleníthetők bizonyos mértékben az alkalmazások.

Egy Enterprise Service Bus implementáció három szempontból jellemezhető:

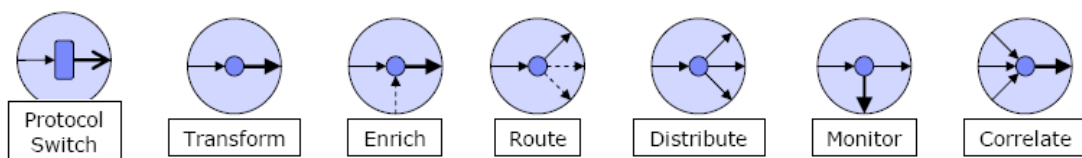
- kommunikációs protokoll és interakciós minta támogatásával
- üzenet modellek létrehozására szolgáló metamodell támogatásával
- mediációs folyamatok létrehozására szolgáló mediációs mintáival

Az Enterprise Service Bus kommunikációs protokoll támogatás segítségével biztosítja a hívók és szolgáltatók közötti kapcsolatfelvételt. A szolgáltatás minőségének

biztosítására támogatja a tranzakcionalitást, azaz műveletsorozatok atomi lefutását (vagy az összes művelet végrehajtódik egy műveletsorozatban, vagy egy sem), illetve a megbízható üzenetküldést. Többféle interakciót támogat a szolgáltatók és a szolgáltatást igénybevevők között. Tipikus interakciós minták a request-reply, one-way vagy speciális esete a publish-subscribe. Request-reply modell esetén a hívó üzenetet küld, és egy választ kap a szolgáltatótól a futás eredményéről, one-way esetben a hívó üzenetére nem érkezik válasz, míg a publish-subscribe modellben a terjesztő által küldött üzeneteket megkapja az összes olyan komponens, amely feliratkozott a terjesztő által küldött üzenetek fogadására. Kommunikációs protokollok terén az Enterprise Service Bus-szal szemben támasztott általános elvárások közé tartoznak a HTTP támogatás (SOAP over HTTP, XML over HTTP), MQ támogatás (SOAP/JMS/MQ, XML/MQ, text/MQ, ...), adapterek örökölt rendszerekhez (legacy system), valamint WS-I, WS-Security biztosítása.

Az üzenet modellek a hívó és a szolgáltató között áramló adatstruktúrák leírását szolgálják. Minden Enterprise Service Bus implementációnak támogatnia kell legalább egy metamodelt. A metamodel segítségével írhatóak le a rendszerben előforduló üzenetek, ezek kapcsolatai, esetleges kényszerek közöttük. Gyakorlatilag a metamodel is egy model, melyet arra használható, hogy az aktuális feladatban használt modellt modellezzük. Metamodellezésre alkalmas eszköz például az XML Schema language. A rendszerben előforduló üzenettípusokat tartalom modellek írják le. Ezekből általában több található a rendszerben. Általában XML Schema segítségével valósítják meg.

A mediációs folyamatok segítségével implementálható a hívó és szolgáltató között áramló üzenetek feldolgozása, irányítása, manipulációja. Intenzíven újrahasznosítható mediációs mintákból (mediation pattern) épülnek fel. Enterprise Service Bus termékekben mediációs primitívek néven hivatkoznak a mediációs mintákra. Tipikus mediációs minták a 7. ábrán láthatóak.



7. ábra Mediációs minták [7]

Ahogy a 7. ábrán látható általános igények a:

- protokollok közötti váltás

- üzenettranszformáció
- üzenet tartalmának bővítése
- üzenet tartalom szerinti irányítása
- üzenet többutas terjesztése
- az áramló üzenetek monitorozása
- üzenetek begyűjtése

1.2 Egyéb szabványok

A fejezet a webes világban elterjedt kliensoldali technológiákat ismerteti, azaz a JavaScript-et és az AJAX-ot. A technológia a megvalósított rendszerben kulcsszerepet játszik, így ismertetése indokolt.

1.2.1 JavaScript

A JavaScript egy objektumorientált szkript nyelv, melynek első megjelenése 1997-re tehető. A nyelv szintaxisa hasonlít a Java nyelvéhez. Eredeti célja HTML oldalak kiterjesztése interaktív funkciókkal. A JavaScriptben írt kód jellemzően böngészőben fut. Az egyetlen széles körben támogatott kliensoldali webes programozási nyelv, ezért nagyon elterjedt.

Alkalmazási területei:

- dinamikus tartalom elhelyezése az oldalon
- HTML elemek tulajdonságainak dinamikus változtatása
- eseménykezelés
- adatok kliensoldali validációja
- böngésző típusának detektálása
- sütik kezelése

Rendelkezik HTML és XML DOM (Document Object Model) implementációval. A DOM egy W3C szabvány, mely egy platform- és nyelvfüggetlen interfészt definiál, mely segítségével programok és szkriptek hozzáférhetnek és manipulálhatják dokumentumok struktúráját, stílusát és tartalmát.

1.2.2 Ajax

Az AJAX az Asynchronous JavaScript And XML rövidítése. A szabvány megjelenése 2005-re tehető. JavaScript-en és HTTP kéréseken alapuló technológia, így

nem nevezhető új programozási nyelvnek, inkább a meglévő szabványok alkalmazásának egy új módja. Segítségével aszinkron kéréseket lehet küldeni és a választ JavaScript segítségével feldolgozni, így az AJAX-ot alkalmazó oldalnak nem kell minden kérés után újraépülnie, csökken az adatforgalom, gyorsabban elérhetőek az oldal funkciói, és így a felhasználói élmény fokozódik. Egy AJAX-os kérés alapja egy XMLHttpRequest objektum. Ennek a W3C által javasolt generálási módján látszik, hogy a nagy böngészőgyártóknak nem sikerült közös szintaxisban megegyezniük.

```
function ajaxFunction()
{
var xmlhttp;
try
{
// Firefox, Opera 8.0+, Safari
xmlhttp=new XMLHttpRequest();
}
catch (e)
{
// Internet Explorer
try
{
xmlhttp=new ActiveXObject("Msxml2.XMLHTTP");
}
catch (e)
{
try
{
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e)
{
alert("Your browser does not support AJAX!");
return false; } } } }
```

A kód alapján látszik, hogy Firefox, Opera és Safari böngészők használata esetén `new XMLHttpRequest()` hívással példányosítható az `XMLHttpRequest` objektum. Internet Explorer-ben és AJAX-ot nem támogató böngészőben a fenti hívás kivételt generál, és a kód `new ActiveXObject("Msxml2.XMLHTTP")` hívással próbálja meg példányosítani az `XMLHttpRequest` objektumot. Amennyiben így sem jön létre az objektum, akkor `new ActiveXObject("Microsoft.XMLHTTP")` Internet Explorer régebbi verzióiban működő hívással próbálkozik a kód. Ha így sem jön létre az objektum, akkor a böngésző egyáltalán nem támogatja az AJAX hívásokat, és ezt hibaüzenettel jelzi a kód.

Miután létrejött egy `XMLHttpRequest` objektum, az életciklusa 5 állapotból áll. Az állapotok listája a következő:

- 0: a kérés még nem inicializálódott
- 1: a kérés inicializálódott
- 2: kérés elküldve
- 3: a kérés feldolgozás alatt
- 4: a kérés feldolgozása elkészült, a válasz megérkezett.

Az XMLHttpRequest objektum állapotváltozásaihoz egy függvény rendelhető, mely minden állapotváltozáskor lefut. Ez a függvény az objektum `onreadystatechange()` metódusával rendelhető hozzá a XMLHttpRequest objektumhoz. Az objektum aktuális állapota a `readyState()` metódussal kérdezhető le. Az objektum `open()` metódusával lehet csatlakozni ahhoz az URL-hez, ahová a kérést küldi az oldal, a `send()` metódussal pedig maga a kérés küldhető el.

Altalában AJAX segítségével meghívott funkció forgatókönyve a következő:

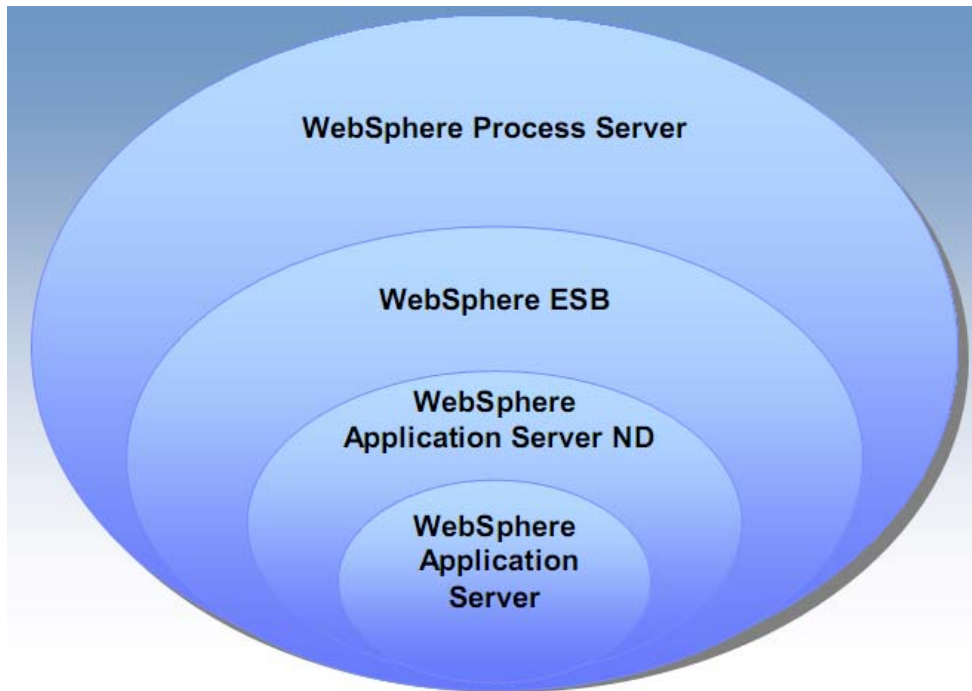
- XMLHttpRequest létrehozása
- hívás paramétereinek kinyerése az oldalból DOM segítségével
- hívás elküldése
- kettes állapotba lépve visszajelzés a felhasználónak, hogy kérése el van küldve, az oldal várakozik a válaszra
- négyes állapotban a válasz feldolgozása, és az oldal frissítése DOM segítségével

1.3 Kapcsolódó IBM termékek

A fejezet célja az IBM WebSphere Enterprise Service Bus és az IBM WebSphere Message Queue alapvető ismertetése.

1.3.1 IBM Websphere Enterprise Service Bus

Az IBM Enterprise Service Bus egyik megvalósítása a Websphere Enterprise Service Bus. A Websphere Enterprise Service Bus a 8. ábrán látható módon a Websphere Application szerverre és a Websphere Application szerver Network Deployment csomagjára épül.



8. ábra Enterprise Service Bus elhelyezkedése [8]

A Websphere Application Server biztosítja a szükséges szolgáltatásminőséget, a J2EE futtató környezetet és az üzenetküldő motort. A Network Deployment csomag biztosítja a nagyvállalati hálózatok számára szükséges képességeket és adottságokat, mint például a fürtözés (clustering), feladatátvétel (failover), nagy rendelkezésre állás és a jó skálázhatóság.

A fürtözésnek két típusa van. Ezek a feladatátvételi fürt és a terheléelosztási fürt. A fürt egy olyan több gépből álló architektúra, melyben a gépek egymással együttműködve és azonos szolgáltatásokat, alkalmazásokat futtatva egyetlen rendszerként, virtuális kiszolgálóként jelennek meg az ügyfelek számára. Feladatátvételi fürt esetén amennyiben egy gép meghibásodik, akkor egy másik veszi át a feladatait és a tárolt adatait. Terheléelosztási fürt esetén egy ütemező osztja szét a fűrthöz beérkező feladatokat a csomópontok között, egyenletes terhelés elérésére törekedve a fürt tagjai között.

A Websphere Application Serverből való származás további előnyei a biztonság, a tranzakciók biztosítása és a központi adminisztráció lehetősége. JMS, HTTP, IIOP és adapter kapcsolódási lehetőségeket biztosít a Websphere Enterprise Service Bus, így a web szolgáltatásokhoz kapcsolódása biztosított. Beépített összekötővel rendelkezik a Websphere Message Queue-hoz, így Websphere MQ hálózatokon keresztül is egyszerűen kommunikálhat. Támogatja az üzenet-, az esemény- és a szolgáltatás-

orientált architektúrákat is. Az üzenetküldésben lehetőség van szinkron, aszinkron, eseményvezérelt és publish/subscribe kommunikációra, így az üzenetküldő mintákkal kellően rugalmasan építhetjük ki rendszerünket. JCA (J2EE Connector Architecture) és WBI (Websphere Business Integration) adaptereken keresztül régebbi alkalmazásainkhoz könnyen hozzáférhetünk, így a fejlesztési időt lerövidíti, hogy ezeket nem kell újraimplementálni. A JCA egy J2EE alkalmazáserver tetszőleges információs rendszerrel való együttműködésének megvalósítását támogató architektúra. Integrált UDDI Service Registry-t tartalmaz, mely megkönnyíti a szolgáltatások és státuszuk nyilvántartását. A fejlesztés gyorsítását és hatékonyságát előre megírt mediációs primitívekkel támogatja, melyek között található – a teljesség igénye nélkül – intelligens üzenet irányítás, XSL transzformáció, loggolás, adatbázisban keresés.

Integráció során szolgáltatás igénylőket és nyújtókat rendelünk egymáshoz, és a közöttük levő mediációs modul segítségével valósítjuk meg a kapcsolat logikáját, azaz átalakítjuk az üzenet formátumát, feltételesen irányítjuk az üzenetet egy vagy több célhoz, vagy éppen bővítjük az üzenet tartalmát. Egy mediációs modul a következő részekből áll:

- exportok, melyek a szolgáltatás igénylőket fel nyújtott interfészeket reprezentálják,
- mediációs folyamat primitívek,
- importok, melyek a szolgáltatókat és interfészeiket reprezentálják,
- egyéni Java komponensek, melyek tetszőleges mediációs primitív funkciókat implementálnak.

Az importok és exportok négyféle kötés valamelyikével rendelkezhetnek. Ezek a következők:

- A JMS kötés támogatja a TCP/IP, SSL, HTTP és HTTPS szállítási protokollokat, valamint képes a WebSphere MQ-val és WebSphere Message Broker-rel kommunikáló rendszerekkel és alkalmazásokkal üzenetet cserélni.
- A Web Services kötés SOAP/HTTP vagy SOAP/JMS protokollokkal küldi az üzeneteket, valamint ismeri a WS-Security és WS-Atomic Transactions szabványokat.

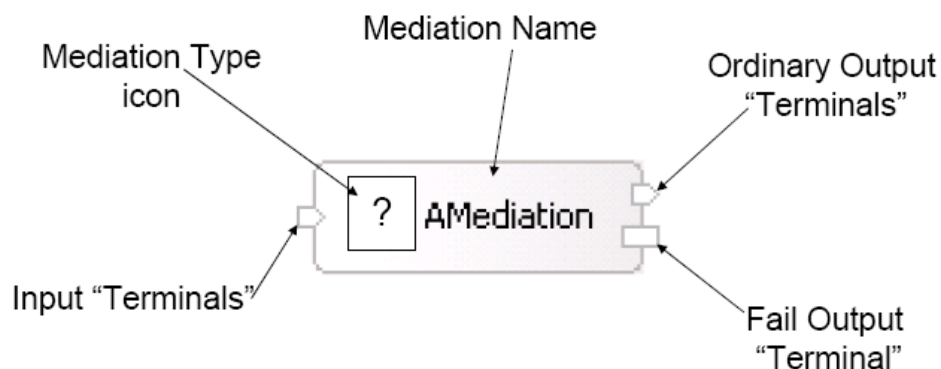
- WebSphere Business Integration adapter segítségével elterjedt nagyvállalati alkalmazások széles körével tudunk kommunikálni. Ide sorolhatóak például SAP, Ariba, PeopleSoft és Siebel termékek.
- A WebSphere Enterprise Service Bus rendelkezik egy alapértelmezett kötés nevű összeköttetéssel is. Ezt modul-modul kommunikáció esetén érdemes használni. A modul-modul összeköttetés segítségével lehet egy folyamat exportján keresztül egy másik importját meghívni közvetlenül. Támogatja mind a szinkron, mind az aszinkron kommunikációt.

Mediációs primitívek

Egy általános mediációs primitív a 9. ábra szerinti grafikus reprezentációval rendelkezik. Rendelkezik:

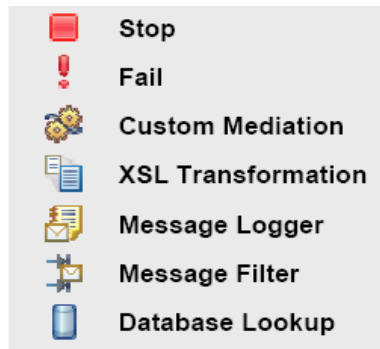
- bemeneti terminállal, ahová a bejövő üzeneteket kapja
- szokásos kimeneti terminállal, ahol az üzenet folytatja útját, ha hiba nélkül feldolgozta a primitív
- hiba kimeneti terminállal, ami hibás futás esetén aktivizálódik,
- valamint az azonosítást megkönnyítendő, középen található a típusára jellemző ikonnal és a nevével.

Speciális primitíveknél tapasztalható eltérés ettől a struktúrától, valamint a bemenetek és a kimenetek száma is igen változó lehet.



9. ábra Mediációs primitív grafikus reprezentációja [9]

A WebSphere Enterprise Service Bus 6.02-es verziójának standard mediációs primitíveinek listája és ikonja a 10. ábrán látható.



10. ábra Standard mediációs primitívek [9]

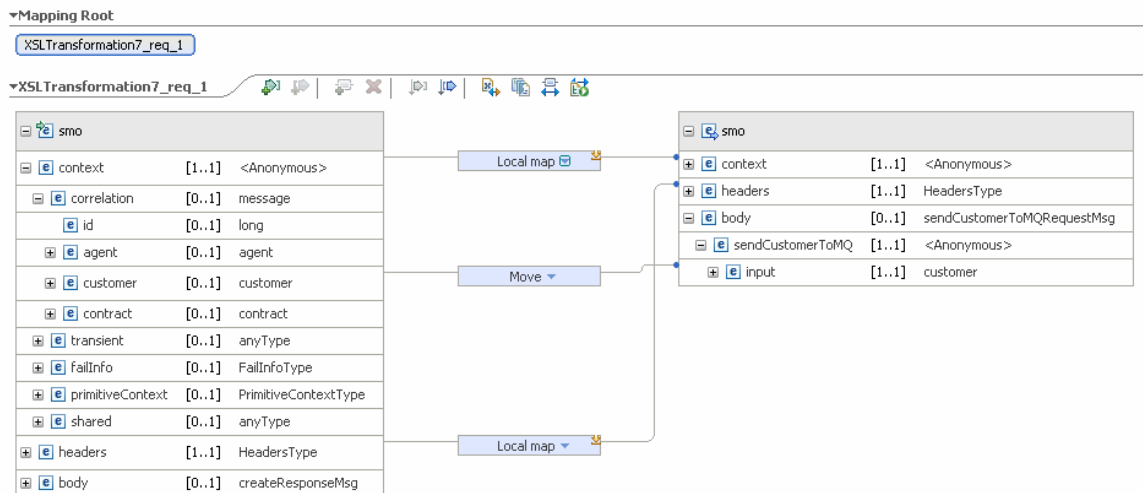
A Stop primitív nem rendelkezik semmiféle kimeneti terminállal. Ha egy üzenet hozzá kerül, akkor az ott megsemmisül kivétel generálása nélkül, azaz bármiféle visszajelzés nélkül. Használata akkor ajánlott, ha egy üzenet feldolgozása a végéhez ért, hiba nélkül végigment a megfelelő futási ágon, és a hívó félnek nincs szüksége semmiféle visszajelzésre. Ha egy szokásos kimenetet nem kötünk össze egy másik primitív bemenetével, az implicit annak felel meg, mintha egy Stop primitívhez kötnénk a kimenetét.

A Fail primitívnek szintén nincs kimenete. Annyiban különbözik a Stop-tól, hogy ha ideér egy üzenet, akkor kivételt generál, valamint létezik egy nem kötelezően megadandó `errorMessage` String típusú tulajdonsága. A kivétel generálásának következtében az üzenet feldolgozása befejeződik, és a hívó kérésére válaszként az `errorMessage`-ben beállított üzenetet kapja. A Fail primitív tipikusan egy hibajelzésre használható egység.

A Custom mediation az üzenet tetszőleges feldolgozására és manipulálására használható. Segítségével a fejlesztő Java nyelven megvalósíthatja azt a feldolgozó egységet, ami a többi primitív felhasználásával nem megoldható.

Az XSL transformation primitív az üzenet tartalmának átalakítását végzi. A 11. ábrán látható grafikus interfészen keresztül lehetőséget nyújt a fejlesztőnek üzenetek transzformációjára. A háttérben a program a grafikuson meghatározott átalakításból XSLT szkriptet készít. A grafikus transzformáció-megadással jelentősen felgyorsul az XSLT szkriptek generálása, valamint beépített tesztelési lehetőség biztosítja a hibázás csökkentését. XSL transzformációval gyakorlatilag tetszőleges formátumúvá és tartalmúvá lehet alakítani a kapott üzenetet. Ennek segítségével lehetséges a szolgáltató interfészének virtualizációja. Tulajdonságainál beállítható a transzformáció XSL alapú leírásának elérhetősége, ami lehet egy URL címen vagy a mediációs modullal azonos

EAR projektben levő JAR fájlban. Megadható továbbá, hogy honnan kezdje a transzformációt, illetve beállítható, hogy a bemenő adatokat validálja-e egy beállított séma szerint. Ha a bemenet ellenőrzése be van állítva igaz-ra, és olyan input érkezik, ami nem felel meg a sémának, akkor kivételt dob, és az üzenet a hiba kimeneten távozik.



11. ábra XSL transzformáció grafikus megadása

A MessageLogger primitív tárolja a rajta áthaladt üzeneteket XML formátumban. Az adatbázis sémáját az IBM előre definiálta. Az adatbázis készítő szkript Cloudscape adatbázishoz készült az ESB 6.0.2-es verzióban, a jelenlegi 6.1.2-es verzióban Apache Derby az alapértelmezett adatbázis. Ha más támogatott adatbázist szeretne az integrátor alkalmazni, akkor azt neki kell létrehoznia és bekonfigurálnia az adott séma alapján. A MessageLogger primitív a felhasznált adatbázisra annak JNDI neve alapján hivatkozik.

A JNDI (Java Naming and Directory Interface) egy Java-ban alkalmazott eljárás objektumok elérésére. Segítségével bejegyzéseket készíthetők adott kontextusokban. A bejegyzések objektumokra (tipikusan külső erőforrásokra) hivatkoznak. A bejegyzés neve alapján később az erőforrások közvetlenül Java kódból elérhetőek.

Ilyen bejegyzéseket az ESB adminisztrációs felületén hozhatunk létre. Az erőforrás elérhetőségén és nevéen kívül meg kell adni annak meghajtóprogramját illetve azt az állományt, mely meghatározza, hogy az üzenet hogyan legyen leképezve az adatbázisba.

A Message Filter primitív az üzenetet a tartalmának megfelelően irányítja tovább. Ebből következik, hogy több kimenettel is rendelkezik, és a tartalom

függvényében dönti el, hogy melyekre továbbítja az üzenetet. A Filter tulajdonsága XPath kifejezés és terminálnév összerendelésekből áll. Jelentése, hogy ha megfelel az üzenet az XPath mintának, akkor a hozzá tartozó terminálon továbbítódik. A szétküldés módja beállítható a `distributionMode` tulajdonságban. Két típusú lehet: vagy az első megfelelő terminálra továbbítódik az üzenet, vagy minden megfelelőre. Ha nincs egyezés, akkor az alapértelmezett terminálra továbbítódik az üzenet. Itt szintén beállítható a gyökér (root) tulajdonságban, hogy az üzenet kiértékelése honnan kezdődjön.

A Database Lookup primitív segítségével adatbázisból nyerhetünk ki információkat az üzenet megadott egyedi atributeuma alapján, és bővíthetjük vele az üzenet tartalmát. A hozzá tartozó megkötések: csak egy táblából tud olvasni, a felhasználónak kell az adatforrást beállítania a primitív számára, a kulcs oszlopnak beállított oszlop csak egyedi értékeket tartalmazhat, valamint az adatoszlopokban található adatoknak Java primitíveknek – azokká alakíthatóaknak – vagy Stringeknek kell lenniük. A `dataSourceJNDIName` és `tableName` tulajdonságoknál az adatforrás helye és a tábla pontos neve adandó meg, a `keyColumnName` és a `keyPath` a táblában a kulcsmezőt és az üzenetben a kulcs helyét megadó XPath kifejezést tartalmazza, a `dataElements`-ben pedig azt lehet megadni, hogy mely mezők milyen típusú értékeit hova mentse az üzenetbe a primitív. Bemeneti formátumellenőrzést itt is lehet kérni, működése analóg az XSL transzformáció primitívénél látottal.

A WebSphere ESB 6.1-es verziója a fentiek mellé három új primitívet vezetett be. Ezek a FanOut, FanIn és a Service Invoke primitívek.

A FanOut primitív a bemenetén kapott üzenetet a kimenetére kötött összes vezetéken elküldi, ezzel párhuzamosítva a feldolgozást. Beállítható, hogy egy üzenetet csak egyszer küldjön el minden vezetéken, vagy működjön iteratív módban, azaz egy XPATH kifejezés minden teljesülésekor küldjön egyet. Az iteratív mód például akkor lehet hasznos, ha adott egy szolgáltatás, mely egy adatstruktúra kezelésére van tervezve, mi viszont ilyen struktúrából többet akarunk küldeni a bemeneten. Erre a mintára szokás "batch processing" néven hivatozni. Ekkor a feladat egy megfelelő XPath kifejezéssel felparaméterezett FanOut primitívvel megoldható. Tapasztalataim szerint a jelenlegi verzióban (WS ESB 6.1.2.002) a FanOut primitív valójában nem párhuzamosítja az üzenet feldolgozását, csupán lefuttat minden ágat sorosan, így teljesítménybeli javulást

nem eredményez a soros végrehajtáshoz képest, csupán „szintaktikai édesítőszerként” használható.

A FanIn primitív egy opcionális gyűjtőobjektumnak tekinthető. Feladata, hogy összegyűjti a hozzá tartozó FanOut-ból érkező üzeneteket, és ha teljesül egy tüzelési feltétele, akkor az utolsóként beérkező üzenetet továbbküldi. Tüzelési feltétel lehet, hogy:

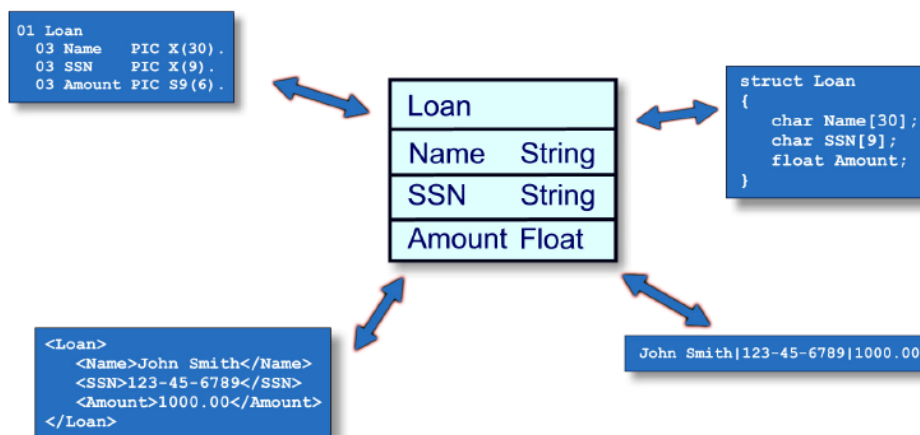
- a beérkező üzenetek száma elért egy bizonyos értéket,
- egy XPath kifejezés teljesült,
- a hozzá tartozó iteratív módban futó FanOut kiküldte az utolsó üzenetét is.

Amennyiben a FanOut primitív már nem küld több üzenetet, és a FanIn utolsó tüzelése után még érkezett üzenet, akkor az inComplete kimenetén jelzi a FanIn, hogy maradtak olyan üzenetek, melyeket nem küldött tovább.

A ServiceInvoke primitív bevezetése lehetőséget teremtett rá, hogy egy szolgáltatást ne csak a folyamat végén lehessen meghívni, hanem annak tetszőleges helyén. A szolgáltatás hívására érkező választ így további szolgáltatások meghívásánál is fel lehet használni, és komplexebb folyamatok építhetők.

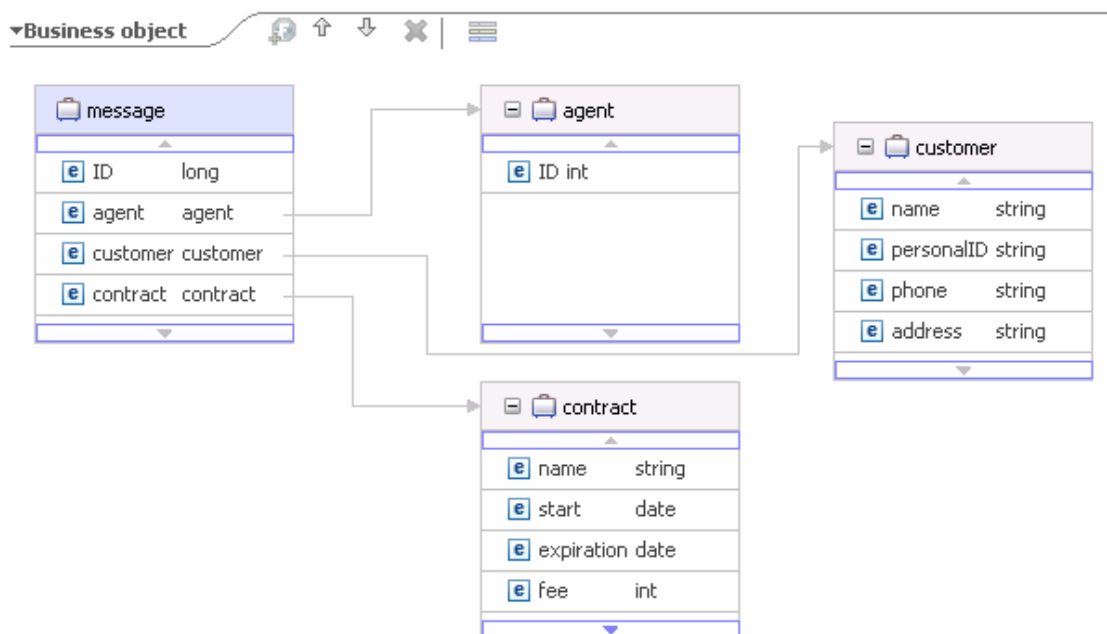
Üzleti Objektumok

Ahogy a 12. ábrán is látható, különböző rendszereknél a megegyező adatok reprezentációjának egyezésére nincs garancia, ezért a WebSphere Enterprise Service Bus egy saját típust, az üzleti objektumot (Business Objects) használja az adatok leírására.



12. ábra Üzleti objektum [9]

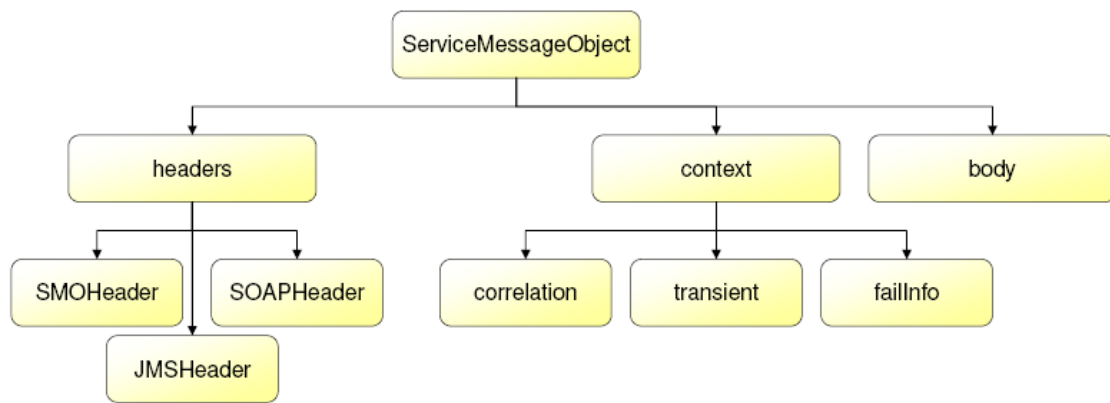
Az üzleti objektumok létrehozása és szerkesztése grafikusán történik a 13. ábrán látható interfészen keresztül. Meghatározhatóak az egyes mezők tartalmára és formátumára vonatkozó kényszerek, mint például a mező minimum és maximum hossza, elhagyhatósága vagy az, hogy milyen értékeket vehet fel. Az üzleti objektum létrehozása közben valójában a formátumát leíró XSD jön létre. Az ESB képességei közé tartozik, hogy az itt megadott kényszereknek megfelelően validálja az egyes folyamatok hívásának bemeneti adatait, és hiba esetén jelzi a kényszer sérülését. Az üzleti objektumok valódi objektumok, azaz a 13. ábrán látható módon más már létező objektumból öröklődhetnek, és egy üzleti objektum tartalmazhat más üzleti objektumokat.



13. ábra Több üzleti objektumot tartalmazó üzleti objektum

Szolgáltatás üzenet objektum

Amennyiben egy kérés érkezik az ESB-hez, akkor létrejön egy szolgáltatás üzenet objektum (Service Message Object - SMO), mely tartalmazza a kérés paramétereit, és a folyamat megfelelő futási ágát végig fogja járni. Ennek az objektumnak az ESB 6.0.2 által használt struktúrája látható a 14. ábrán. Egy SMO tartalmaz fejléc-, kontextus-, hiba- és törzsinformációkat. Az egyes mezők alkalmazás programozói interfészen keresztül elérhetőek.



14. ábra Szolgáltatás üzenet objektum hierarchiája [9]

A törzs rész tartalma és struktúrája változhat, miközben az ESB-ben található folyamaton halad végig az SMO. Tartalma tipikusan egy meghívandó szolgáltatás hívási paraméterei, vagy ha egy szolgáltatásból érkezik az SMO, akkor a szolgáltatás által küldött válasz.

A kontextus/korreláció mező tartalma az integrátor által megadott üzleti objektum. A szolgáltatás hívó folyamatban állítódik be az értéke, és a válasz folyamatban is hozzáférhető lesz. Gyakori tervezési minta, hogy amennyiben egy kérést több szolgáltatóhoz is el kell juttatni, akkor a rendszer fejlesztője az eredeti kérés paramétereit lementi a korreláció mezőbe, és az egyes hívásoknál XSL transzformációval onnan helyezi az SMO törzs részébe a szükséges mezőket. Látható, hogy ha nem tartalmazná az SMO korreláció mezője az eredeti kérést, mely eredetileg a törzsben található, akkor az első meghívott szolgáltatás válasza felülírná az eredeti kérés paramétereit, és jó eséllyel a struktúráját is, így újabb szolgáltatás már nem lenne hívható a folyamatban a kezdeti adatokkal.

A kontextus/tranziens mező jellegét tekintve nagyon hasonlít a korrelációra. Szintén egy üzleti objektumot tárol, és szintén ideiglenes adatmentési területként használható. Annyi különbség van viselkedésében, hogy tartalma a válaszfolyamatban már nem érhető el. Létezősége van, hiszen ezzel a fejlesztő nyert plusz egy ideiglenes tárolóhelyet, ahová adatait mentheti a hívási folyamatban, azzal pedig, hogy a válaszfolyamatban már nem tárolódik, a rendszernek erőforrást spórol.

A fejlécek mezőkben a feldolgozáshoz szükséges jellemző fejlécinformációk találhatóak. A fejlesztő nem szokta ezeket a mezőket szerkeszteni, futás közben automatikusan kapnak értéket.

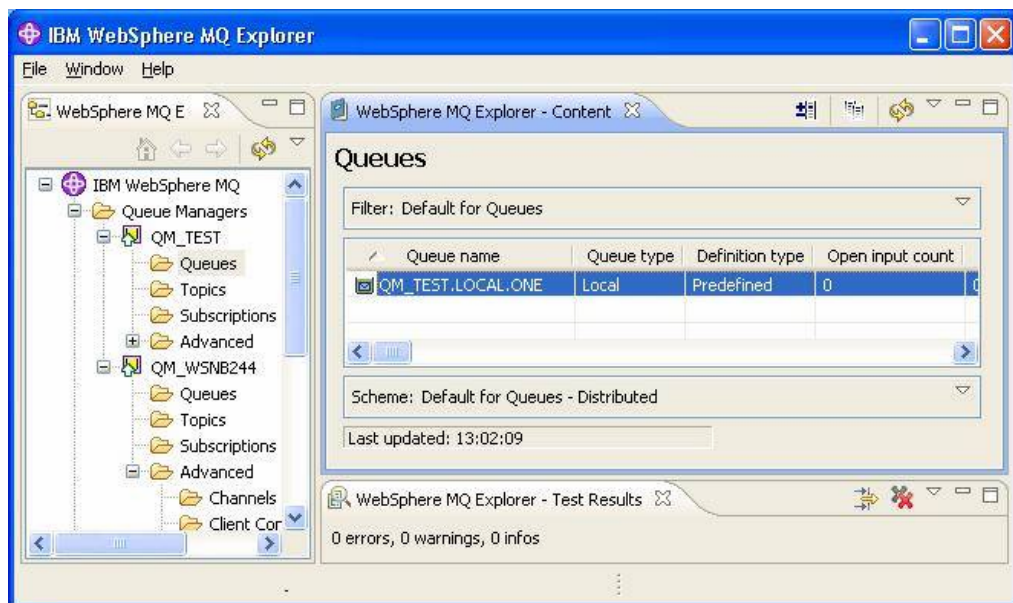
A kontextus/hibainformáció akkor kap értéket, ha egy primitív kivételt dob. Ekkor innen kiolvasható a hiba oka, és ez alapján szükség szerint módosítható a folyamat.

Az ESB 6.1-es változatában bevezettek egy új kontextust, ez pedig a Shared Context. Mivel a folyamatban megjelent a párhuzamosítás, ezért felmerült az az igény, hogy a különböző ágakon futó SMO-k tudjanak egy közös területre írni. Ezt a funkciót hivatott ez az új terület megoldani.

1.3.2 IBM WebSphere MQ

Az IBM WebSphere Message Queue (MQ) az IBM zárt üzenetküldő technológiája. Segítségével üzenetsorokat lehet létrehozni, melyekre számos támogatott programnyelvhez tartozó API-n keresztül üzeneteket lehet elhelyezni vagy levenni.

MQ üzenetsorokhoz hozzá lehet férni mind JMS (Java Message Service), mind MQI (Message Queue Interface) interfészen keresztül. Támogatja többek között az üzenetek biztos megérkezését, az üzenetek sorrendezését és a duplikált üzenetek szűrését. Alkalmas eszköz különböző nyelven íródott alkalmazások üzenetsorokkal történő interakciójának megvalósítására, ahol az üzenetek biztos és sorrendhelyes megérkezése követelmény. A 15. ábrán látható Eclipse alapú grafikus interfész segítségével biztosított az üzenetsorok menedzselése, konfigurálása. Az IBM WebSphere MQ támogatja a publish/subscribe üzenetküldést.



15. ábra IBM WebSphere MQ Explorer

2 Rendszertervezés

A fejezet a rendszer tervezésének dokumentációja. Tartalmazza a rendszer általános leírását, a szerepkörök és műveleteik definiálását, a felhasznált kliens koncepciójának eldöntését, az adatfrissítés mechanizmusának kiválasztását valamint autentikációs problémák megoldását. Az alapvető koncepció meghatározása után a rendszerben előforduló forгатókönyvek dokumentációja kerül ismertetésre, s végül egy külön fejezet tér ki a rendszer felépítéséhez szükséges komponensekre és ezek hibájának következményeire és kezelésére.

2.1 A rendszer funcionális leírása

A megvalósítandó e-Business rendszer célja egy koordinátor cég, annak leányvállalatai és beszállítói informatikai rendszere közti interakció megkönnyítése.

A koordinátor cég ügyintézőkből áll. Az ügyintézők feladata a leányvállalatok árukészletének szinten tartása. A rendszer a bejelentkezett ügyintézők számára egy felületet biztosít, melyen keresztül monitorozhatják a leányvállalataik raktárkészletét leíró paramétereket.

A megfigyelt adatok függvényében az ügyintézők rendeléseket adnak le a rendszeren keresztül a beszállítóknak a rendelendő termék azonosítójának, mennyiségének, szállítási címének és saját azonosítójuk megadásával. A rendszer az így megadott rendelést a későbbi visszakereshetőség és az esetleges felelősségrevonás érdekében naplózza, majd a megfelelő beszállítóhoz eljuttatja. Amennyiben a beszállító technikai problémák miatt többszöri újrapróbálkozás után sem érhető el, vagy a rendelés valamilyen okból nem teljesíthető, akkor a rendszer az ügyintézőt erről tájékoztatja, és a rendelést elveti. Sikeres rendelés esetén a beszállító egy rendelésazonosító visszaküldésével nyugtázza a rendelést.

Az ügyintézők hatékony munkavégzésének támogatása érdekében a rendszer támogatja a rendelések lekérdezését is az ügyintéző által megadott intervallumon. Egy lekérdezés folyamán az ügyintéző egy táblázatot kap a megadott intervallumon belül leadott rendelések paramétereiről. A leányvállalat készletének adataiból és a folyamatban levő rendelésekből az ügyintéző kellő információt kap, hogy eldönthesse, szükséges-e újabb rendelések leadása, vagy a függőben levő szállítmányok fedezni fogják a leányvállalatok szükségleteit.

Amennyiben az ügyintézőnek rendelnie kell, akkor a megfelelő termék kódjára rákeresve kilistáztatja az egyes beszállítóknál a megadott kódon kapható termékeket. A rendszer biztosítja, hogy a termékről az ügyintéző minden fontosabb adathoz hozzáférjen, így a termék kódjának megadásával az ügyintéző megtekintheti, hogy az egyes beszállítóknál mi az aktuális ára, és jelenleg mennyi van belőle készleten. Ezen információk alapján az ügyintéző kiválaszthatja azt a beszállítót, amely rendelkezik a rendelő mennyiséggel, és megfelelő áron tudja biztosítani az terméket.

Minden beszállító rendszerének biztosítania kell a központi rendszer felé egy szolgáltatást, melyen keresztül az ügyintézők árut rendelhetnek tőlük. A szolgáltatásnak sikeres rendelés esetén egy rendelésazonosítóval kell visszajelezni a rendelés elfogadását, hiba esetén pedig hibaüzenetet kell visszaküldenie. A beszállítók rendszerének a központi rendszer által biztosított interfészen keresztül készlet- illetve árváltozásait folyamatosan jelezniük kell, biztosítva ezzel, hogy az ügyintézők a lehető leghamarabb a legfrissebb adatokat láthassák készletükről.

A leányvállalatok alkalmazottainak feladata szintén kettős: jelezniük kell a központi rendszer felé, amennyiben megváltozott a készletük – például eladás hatására – valamint azt, ha egy beszállító teljesítette az ügyintéző által leadott rendelést.

2.2 A rendszer működésének alapvető koncepciói

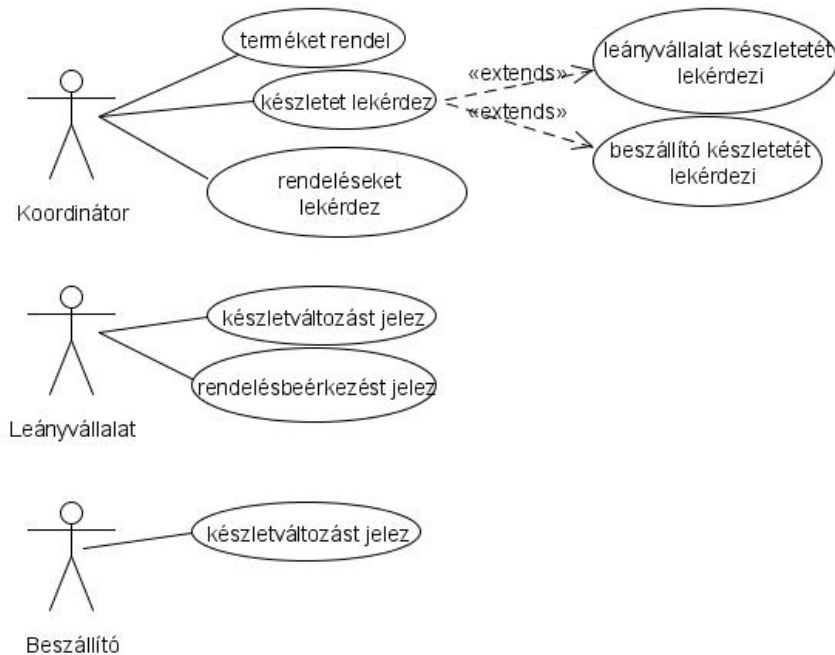
A fejezet a rendszer alapvető koncepcióit tartalmazza, így a szerepkörök meghatározását, a felhasznált kliens típusát, az adatfrissítés mechanizmusát és a kérések autentikációját.

2.2.1 Szerepkörök

Mint minden többfelhasználós rendszert, ezt is számos felhasználó használja különböző jogosultságokkal. A felhasználók tipikusan különböző csoportokba oszthatóak aszerint, hogy milyen műveletek elvégzésére van jogosultságuk a rendszerben. Ezeket a csoportokat szerepköröknek nevezzük. Jelen rendszer felhasználói három szerepkörbe tartoznak. Ezek a szerepkörök a következők:

- koordinátor cég ügyintézője
- beszállítói cég alkalmazottja
- leányvállalat dolgozója

A három szerepkörhöz tartozó felhasználók különböző tevékenységeket végezhetnek a rendszerben, melyek a 16. ábrán látható használati eset diagramokon tekinthetők meg.



16. ábra Szerepkörök és elvégezhető műveleteik

Az ábra alapján a koordinátor cég ügyintézői a rendszeren keresztül rendelhetnek, lekérdezhetik beszállítóik és leányvállalataik készletét valamint adott időintervallumon belüli rendeléseiket. A leányvállalat alkalmazottai készletváltozást és rendelésbeérkezést jelezhetnek, a beszállítók pedig készletük változását. A készletváltozás lehet az áru egységárára vagy annak mennyiségére vonatkozó változás. A diagram előrevetíti azt, hogy a változásokról melyik félnek kell tájékoztatnia a másikat. Ennek magyarázata a 2.2.3 fejezetben található.

A rendszerben történő műveletvégzéshez természetesen a felhasználóknak autentikálni kell magukat, mivel nem megengedhető egy rendszeren kívüli személynek, hogy bármiféle változást jelezzon, vagy akár raktárkészletet kérdezzen le, így bevezethető általánosabb szerepkörként egy "felhasználó" szerepkör, melynek autentikáció az elvégezhető művelete. A komplett rendszerhez természetesen szükség van még egy admin szerepkörre, akinek feladata a felhasználói fiókok menedzselése.

2.2.2 Felhasznált kliens

A felhasználók által használt kliens lehet vékony és vastag kliens alkalmazás is. A vékony kliens használatának előnye, hogy nincs szükség semmiféle telepítésre, és a

platformfüggetlenséget is biztosítja. Mivel a szükséges kliensoldali logika megvalósítható 1.2.1 és 1.2.2 fejezetben említett JavaScript és AJAX segítségével, valamint a felhasználói élmény is jól közelíti egy vastag kliens alkalmazását, ezért ez egy kellően rugalmas és könnyen terjeszthető alternatívának számít.

Vastag kliens alkalmazás esetén a telepítés nem spórolható meg, ráadásul platformfüggő nyelv használata esetén vagy több változatot kell a kliensből csinálni, vagy le kell mondanunk a több platformon való futtathatóságról. A fentiek alapján a vékony kliens a megfelelő választás.

Az ügyintéző a rendszer használatakor a következő lépéssorozat alapján dolgozik:

- ellenőrzi a leányvállalatok készletét
- amennyiben valamelyik termékből kevés van egy leányvállalatnál, megnézi, hogy van-e rendelés, amely oda tart, de még nem érkezett meg
- ha van, akkor visszatér az első lépéshez, egyébként rákeres a termék nevére, és kap egy táblázatot a beszállítók készletéről, amely tartalmazza a termék árát az adott beszállítónál és az ott rendelkezésre álló mennyiséget.
- a megfelelőt kiválasztva elküldi a rendelést

A fenti szekvencia alapján a 17. ábrán látható interfész megfelel a követelményeknek.

Készlet

Budapesti kirendeltség			Kecskeméti kirendeltség		
Termékkód	Terméknév	Darabszám	Termékkód	Terméknév	Darabszám
1243	100-as Fűrő	100	1243	100-as Fűrő	100
1324243	10480-as Fűrő	0	1324243	10480-as Fűrő	0
13243	1 GB DDR Samsung memória 10		13243	1 GB DDR Samsung memória 10	

Rendelések

Szűrési feltétel
 Rendelési időpont kezdete: Rendelési időpont vége:

Találatok

Rendelésazonosító	Rendelés dátuma	Rendelte	Termékkód	Terméknév	Darabszám	Egységár	Beszállító	Szállítási cím	Szállítva
1234234243	2009.01.12.	TothGizi	14243	1000-es fűrő	100	5423 Ft	Obi	Teszt utca 1.	igen
1234243	2009.01.21.	TothBela	14243	1000-es fűrő	100	5423 Ft	Obi	Teszt út 2.	nem

Termék rendelése:

Keresés:
 Megrendelendő termék kódja:

Találatok:

Termékkód	Terméknév	Darabszám	Egységár	Beszállító	Rendelendő mennyiség	Szállítási cím	Rendelés
1243	100-as Fűrő	100	5423 Ft	Obi	<input type="text"/>	Budapesti kirendeltség	<input type="button" value="Rendelés"/>
1243	100-as Fűrő	10000	5600 Ft	Ikea	<input type="text"/>	Budapesti kirendeltség	<input type="button" value="Rendelés"/>

17. ábra Felhasználói felület

A felülről az összetartozó részeket keret fogja körül a könnyebb tájékozódás érdekében. Az összetartozó részek három csoportra oszthatóak, ez a leányvállalatok készlete, rendelések, és a termék rendelése részleg. Sorrendjüket az ügyintéző által végzett munka fázisai alapján alakítottam ki.

Tervezői döntés, hogy a termék rendelésénél a keresés gombnyomás hatására történjen, vagy a beíráskor folyamatosan küldjön az oldal kéréseket. Az automatikus küldés jelentős terheléstöbbletet jelent a rendszernek, hiszen az adatbázisnak minden gombnyomásra egy keresést kell végeznie. Cserébe annyi többlétszolgáltatást kap a felhasználó, hogy nem kell a "keresés" gombra kattintania. A nyereség láthatóan meglehetősen kicsi, viszont az költsége magas, így a keresés gombot választottam.

2.2.3 Adatok frissítése

A rendszer egyik fontos feladata, hogy a beszállítók készletének adatait a koordinátor céghez eljuttassa. Ez történhet eseményvezérelten a beszállító által kezdeményezve és idővezérelten a koordinátor cég által kezdeményezve. Idővezérelt esetben a koordinátor cég gyakorlatilag periodikusan lekérdezi a beszállítók készletének azt a részalmozását, amire az ügyintézőknek éppen szükségük van. A módszer hátránya, hogy pazarlóan bánik az erőforrásokkal, mivel akkor is kérdegeti és feldolgozza az raktárkészlet adatokat, amikor nem történik a rendszerben változás. A lekérdezési intervallum növelésével lehet csökkenteni a terhelést, viszont ekkor az ügyintézők egyre később értesülnek a változásokról. A rendszer jobb hatásfokkal működik, ha eseményvezérelten, a beszállító által indítva értesül a koordinátor cég a változásokról. Ekkor, ha egy termék valamely paramétere megváltozik, akkor azt jelzi a beszállító a koordinátor cég felé, így csak akkor folyik közöttük kommunikáció, ha arra valóban szükség van.

A fent leírtak érvényesek a leányvállalat készletének és a rendelések beérkezésének változásairól történő értesülésekre is, ezért itt szintén az eseményvezérelt esetet választottam.

Az ügyintéző bejelentkezése után az oldalnak el kell érnie azt, hogy a beszállítók, leányvállalatok és rendelések változásairól értesüljön. Ez megvalósítható úgy, hogy periodikusan lekérdezi azoknak a termékeknek és rendeléseknek az adatait, melyekkel az ügyintéző éppen dolgozik. Ez nagy felhasználószám és ritka változás esetén nagymértékű felesleges forgalmat generál. Szébb megoldás, ha a belépett felhasználók oldalai AJAX kérések segítségével figyelnek egy közös csatornát, melyen

keresztül a változásokról értesülhetnek. Az oldal a csatornán keresztül érkező adatok alapján el tudja dönteni, hogy a változás érinti-e a felhasználó által látott nézetet. Amennyiben igen, akkor ha szükséges, egy lekérdezéssel lekéri az új adatokat, és a válasz alapján aktualizálja a felületet, ha pedig a csatornán megtalálható minden szükséges adat, akkor egyszerűen megjeleníti azt. A közös csatornák megvalósíthatóak publish/subscribe modell segítségével.

Publish/subscribe terminológiában a közös csatornát témának nevezik, míg a figyelését a "témára való feliratkozásnak". A különböző témák esetén különböző adatokat kell szolgáltatni az ügyfelek számára.

A leányvállalatok minden adatát látnia kell az ügyintézőnek, így a leányvállalatok változásai témára a megváltozott termék minden adata rákerülhet, így további lekérdezés nélkül frissíthető a leányvállalatok rész.

A rendelések változásai témán első ránézésre elengedő lenne a rendelésazonosítót kiküldeni, és a kliensnél ha látszik az adott rendelésazonosító, akkor csupán a státuszát kellene megváltoztatnia. Ez azonban nem jó megoldás, hiszen ha egy új rendelés kerül a rendszerbe, akkor annak rendelésazonosítója még sehol sem szerepel, így egy kliens sem érezné úgy, hogy frissítenie kell a rendelések táblázatát. Ez alapján azt az információt kell kiküldeni, hogy mikori dátumú rendelés változott, és amennyiben az ügyintéző által figyelt intervallumba esik, akkor az oldalnak frissülnie kell. Ekkor küld egy kérést, mellyel lekérdezi újra a rendeléseket a felhasználó által megadott intervallumon, és a lekérdezés eredményét megjeleníti. Az előzőek mellett megvalósítható egy olyan rendszer is, ahol a dátum és rendelésazonosító együtt kerül a rendelésváltozások témára, és a dátum alapján azt dönti el az oldal, hogy érdekes-e számára a változás, és ha igen, akkor csak a kapott rendelésazonosítóra keres rá. A második módszer kisebb sávszélességet igényel, viszont a kliensoldali logika bonyolultabb, ezért az elsőt választottam.

A beszállítók készletének változásainak frissítése több módon is megvalósítható. Lehetséges úgy, hogy a beszállítók változásai témán a termék kódja és beszállítója van egy üzenetben. Ha az ügyintézőnél a keresés szekció találatok részében szerepel a termék, akkor az oldal egy AJAX kérés segítségével lekéri annak adatait, és frissíti tartalmát a válasz alapján. Másik megoldás, hogy a változás adata is benne van az üzenetben, azaz az új ár vagy az új mennyiség. Az első esetben az üzenet kisebb lesz, így valamivel kevesebb hálózati forgalmat generál a módszer. A módszer nagy hátránya, hogy az összes kliens, akit érint a változás, egyszerre küld egy kérést, hogy

lekérdezze a változás paramétereit, így az adatbázis a változások után egyszerre kapja a kiszolgáló kéréseket. Ezeket a szempontokat nézve a tervezőnek a kis mértékű hálózati forgalomtöbblet és az adatbázisterhelés növelése között kell döntenie. Ez alapján a második módszert választottam.

2.2.4 AJAX kérések autentikációja

Az ügyintézők által használt felület AJAX kérések segítségével frissíti magát, illetve a rendeléseket is AJAX segítségével küldi az oldal. A kéréseket azonosítani kell, hiszen nem megengedhető, hogy bárki tudjon ilyen kéréseket küldeni, illetve az sem, hogy később ne lehessen visszakeresni például az egyes rendelések feladóit. Az azonosításhoz szükséges információknak az oldal forrásában megtalálhatónak kell lennie, mivel a JavaScript kliensoldali technológia. Az alábbi azonosítási lehetőségek lehetségesek:

- kéréssel együtt felhasználónév elküldése
- kéréssel együtt felhasználónév és jelszó elküldése
- azonosító token küldése

Amennyiben a kéréssel együtt elküldi a rendszer a belépett felhasználó azonosítóját, akkor triviálisan azonosítható a kérés azonosítója. A probléma a módszerrel az, hogy rendkívül nagy biztonsági rést nyit a rendszerben, hiszen egy felhasználó felhasználói neve viszonylag könnyen kiszivároghat.

Nagyobb védelmet nyújt ennél, ha a felhasználónév és a jelszó is elküldésre kerül a kéréssel együtt, ez azonban szintén komoly biztonsági rés, hiszen ekkor a felhasználónevet és jelszót a kliensoldalon tárolni kell, és aki hozzáfér a géphez, az könnyedén megszerezheti így a belépéshez szükséges felhasználónév és jelszó párost. Amennyiben titkosítatlan csatornán utazik a kérés, akkor egy lehallgató támadó könnyedén hozzáférhet az autentikációs adatokhoz.

Azonosító tokenes azonosítás esetén a rendszer belépéskor generál a felhasználónak egy véletlen számot, melyet adatbázisba ment. Későbbiekben a felhasználó ezt a számot minden kéréshez (például rendeléshez) elküldi. Az azonosítószám alapján a kliens azonosítható, és annak esetleges nyilvánosságrakerülése esetén is legfeljebb addig használható, amíg a felhasználó újra be nem lép, hiszen minden belépéskor új generálódik, így a régi érvénytelenné válik. A módszer következménye, hogy egy felhasználó egyszerre csak egy helyen lehet bejelentkezve. Az azonosító lehallgatása ellen ideális védelem a csatorna titkosítása, így a támadó csak

akkor tud hozzáférni az azonosítótokenhez, amennyiben a géphez is hozzáfér, s a token is legfeljebb addig tudja használni amíg újra be nem lép a felhasználó.

2.3 A rendszerben előforduló forgatókönyvek

A rendszer használata során jól elkülöníthető művelet sorokat végeznek a felhasználók. Ezeket a művelet sorokat nevezhetjük forgatókönyveknek. A fejezet ezen forgatókönyvek leírását tartalmazza.

2.3.1 Ügyintéző bejelentkezése

Az ügyintéző bejelentkezése az autentikációs folyamattal kezdődik, ahol megadja felhasználónevét és jelszavát. Sikertelen autentikáció esetén a rendszer természetesen újra bekéri a felhasználótól ezeket az adatokat, egyéb esetben a 2.4 fejezetben említett azonosítótokent generál, amit adatbázisba ment. A rendszer ezután automatikusan a 17. ábrán látható oldalra irányítja a belépett felhasználót. A rendszer elhelyezi az oldal forrásában az AJAX kérések azonosításához szükséges token, és AJAX segítségével lekérdezi a leányvállalatok készletét.

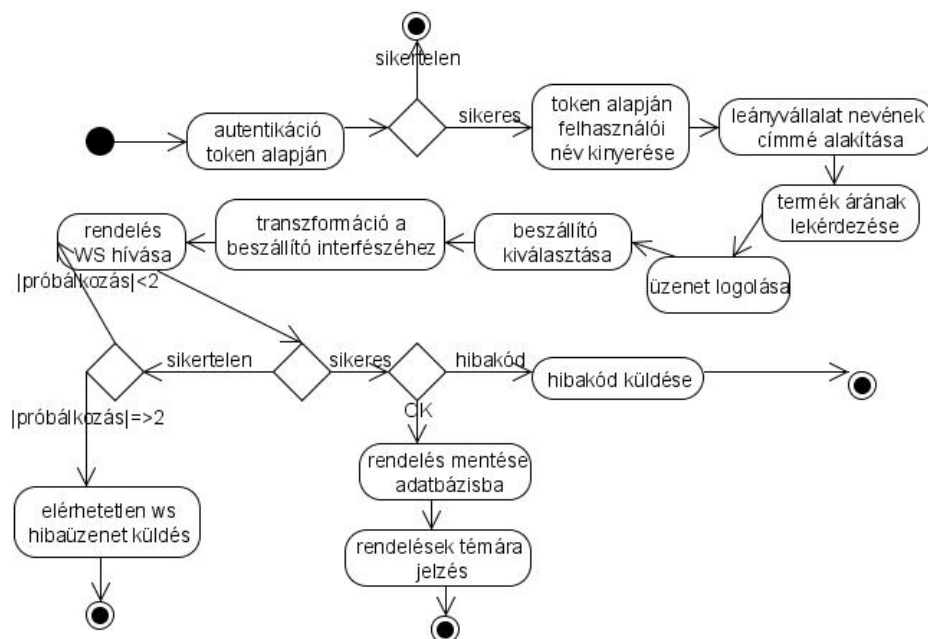


18. ábra Ügyintéző bejelentkezése

2.3.2 Rendelés leadása

Az ügyintéző a felhasználói felületen kiválasztja a rendelendő terméket, beszállítóját, a megrendelendő mennyiséget és a leányvállalat nevét, ahová a szállítást kéri. A rendelés adataival együtt az oldal a kéréssel együtt elküldi az azonosítótokent is, melyet első lépésben a rendszer ellenőriz. A rendelés folyamata innentől a 19. ábrán

nyomonkövethető. Amennyiben létezik ilyen token az adatbázisban, akkor a hozzá tartozó felhasználónevet kikeresi a rendszer. Az ügyintéző a leányvállalat nevét adta meg a rendelésnél, a beszállítót viszont a szállítási cím érdekli, ezért a rendszer adatbázislekérdezés segítségével átalakítja a nevet címmé a következő lépésben. A rendszer lekérdezi a rendelt termék árát, majd a kérést logolja. A rendszer a beszállító paraméter alapján kiválasztja, hogy melyik szolgáltatást kell meghívnia, és a paramétereket a beszállító interfészéhez igazítja. A rendszer korlátozott számúszor újrapróbálkozik a hívással, elérhetlenség esetén hibáüzenettel jelzi azt. Amennyiben sikerült a hívás, két eset lehetséges: hibakódot küld vissza a beszállítói szolgáltatás vagy rendelésazonosítót. Hibakód küldésének oka lehet például, hogy a termékből a rendelés küldése közben elfogyott annyi, hogy már nem lehet a rendelést teljesíteni. Rendelésazonosító küldése esetén a rendelést a beszállító elfogadta. Ekkor a rendszer menti a rendelések közé annak adatait, és a rendelések témára küld egy jelzést, hogy új rendelés került a rendszerbe.



19. ábra Új rendelés leadása

A folyamat során számos adatbázishozzáférés történik, melynek sikertelensége esetén a rendszer természetesen hibáüzenettel jelzi azt. Ezek az ágak a diagram áttekinthetőségének érdekében nem szerepelnek.

2.3.3 Rendelések lekérdezése

A felhasználó két időpont megadásával kijelölhet egy intervallumot, amelyen belüli rendeléseket ki akar listázni. A folyamat az autentikációs lépés után egy egyszerű adatbázislekérdezéssel lekéri a megadott időintervallumon belüli rendeléseket, és válaszban visszaadja azt. A válasz formátumának egy kellően rugalmas, JavaScript segítségével feldolgozható struktúrának kell lennie, mivel a lekérdezés eredménye változó számú termék lehet. Az XML rendelkezik ezekkel a tulajdonságokkal, mivel rugalmas, és JavaScripthez létezik DOM értelmező, mellyel kliensoldalon feldolgozható, ezért a válasz formátumaként XML-t választottam.

2.3.4 Beszállítói készlet változás

A beszállító a rendszer által biztosított szolgáltatáson keresztül jelzi raktárkészletének vagy egy terméke árának változását. Sikeres autentikációs lépés után a beszállító által küldött adatokat a rendszer adatbázisba menti, majd a beszállítói változások témára jelzi, hogy a beszállító adott kódú terméke adott árra vagy mennyiségre változott. Az ügyintézők által figyelt oldalak igény szerint frissítik magukat a kapott üzenet alapján.

2.3.5 Rendelés beérkezése

Amennyiben egy rendelés beérkezik egy leányvállalathoz, akkor a leányvállalat ezt jelzi a rendszernek a beszállító és a rendelésazonosító megadásával. A beszállító nevére azért van szükség, mert nem garantálja semmi, hogy két különböző beszállító nem adja ugyanazt a rendelésazonosítót két különböző szállítmánynak. Autentikáció után a rendszer megváltoztatja a rendelés státuszát, majd a rendelés dátumát elküldi a rendelések változása témára. Ezzel jelezi, hogy egy ekkori dátumú rendelés megváltozott, amelyik kliensnek szükséges, az frissítse magát. A hívást a 2.3.6 fejezetben ismertetett leányvállalat készletének változása hívás követi, amin a leányvállalat az új készletét jelezheti a koordinátorcég felé.

2.3.6 Leányvállalat készletének változása

A leányvállalat készletének változása analóg a 2.3.4 fejezetben ismertetett beszállító készletének változásával, azzal a különbséggel, hogy a leányvállalat csak a raktárkészletét jelzi a koordinátor cég számára, az ár információt nem.

2.4 Rendszer komponensei és kapcsolata

A fejezet tartalmazza a rendszer megvalósításához szükséges komponensek kiválasztását valamint az azok meghibásodásából származó következményeket és azok kezelését.

2.4.1 A rendszer komponenseinek kiválasztása

Az aktivitás diagramokon látható, hogy a rendszerben gyakran szükség van az üzenetek átalakítására, útvonalválasztására, naplózására. Ezen kívül még több viszonylag független szervezet között kell megvalósítani egy rendszert, melyek statikusságára semmi garancia nincs, így általános igényként felmerül a rugalmas, költséghatékony változtathatóság. Ezek az igények egy ESB komponens beiktatását indokolják a rendszerbe.

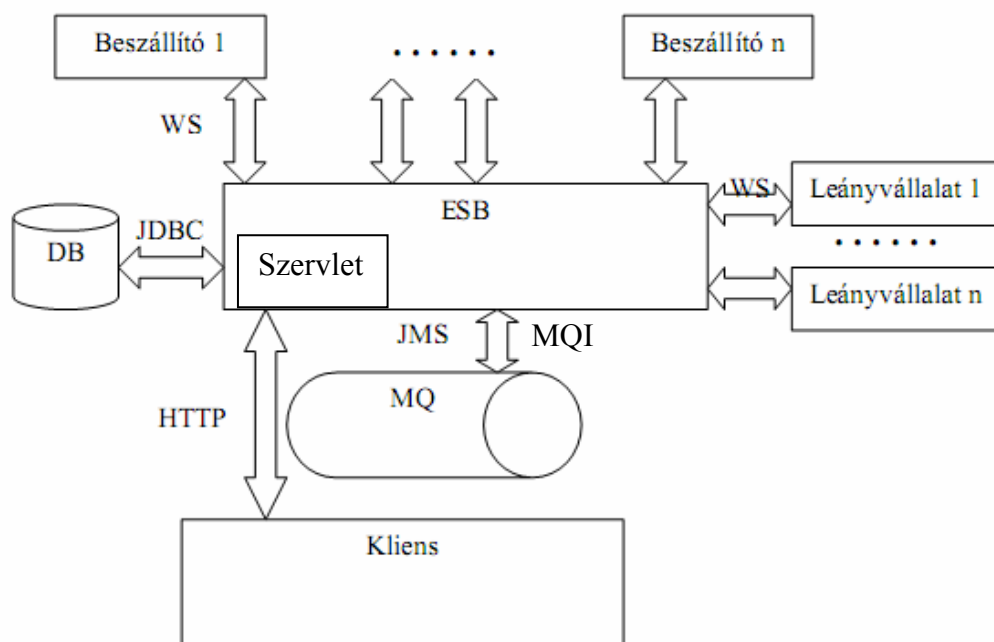
A rendelések, és a készletek tárolásához szükség van egy perzisztens tárra. Erre hatékony megoldásként adatbázis ajánlott, mely az ESB-hez csatlakozik, és közöttük JDBC segítségével történik az adatcsere.

A publish/subscribe üzenetküldés szükséges a belépett felhasználók kliensei számára, hiszen a 2.2.3 alapján így értesülnek a változásokról. Mivel az MQ rendelkezik ezzel az üzenetterjesztő képességgel, ezért bevezetése indokolt. Az ESB-vel JMS és MQI segítségével kommunikál.

A kliens AJAX kéréseket küld a rendszernek HTTP felett, és így kell elérnie az MQ üzenetsorokat és témákat. Ez egy feltetepített szervlet segítségével megvalósítható. A szervlet az ESB-ben fut.

A beszállítók és a leányvállalatok a kellő rugalmasság és függetlenség elérése miatt web szolgáltatásokon keresztül kommunikálnak az ESB-vel. A web szolgáltatások alkalmazásának köszönhetően nincs megkötés az egyes rendszerek platformjaira, valamint a rendszereik implementációját is szükség szerint cserélgethetik a szolgáltatók.

A rendszer architektúrája a 20. ábrán tekinthető meg.



20. ábra Komponensek kapcsolata

2.4.2 A komponensek hibájának következménye, kezelése

A rendszert fel kell készíteni a különböző komponensek kiesésének kezelésére.

Ezek a komponensek a következők:

- adatbázis
- ESB
- MQ
- beszállítók rendszere

Amennyiben az adatbázis nem elérhető, új felhasználó már nem tud belépni a rendszerbe, mivel a felhasználónevekhez és jelszavakhoz már nem fér hozzá az ESB. A belépett felhasználók újabb lekérdezéseket nem tudnak küldeni, mivel azonosító tokenjük szintén az adatbázisban van tárolva, így az autentikációs lépésnél minden beérkező kérés el lesz utasítva az adatbázis elérhetetlensége miatt. A rendszerben problémát jelent, ha az adatbázis akkor válik elérhetetlenné, amikor a beszállító visszajelezte a rendelést, de az ESB már nem tudja a rendelések közé menteni. Ekkor keletkezik egy olyan rendelés, amiről a koordinátor cég "nem tud". Erre az esetre a megoldás az, hogy ilyenkor az ESB a fájlrendszerbe lementi a rendelés adatait, és amikor az adatbázis újra elérhető, a fájlokban tárolt adatokat az adatbázisba kell menteni. Ez történhet akár kézi, akár automatizált módon szkriptekkel. A kézi megoldás

hátránya lassúságán kívül, hogy a hibázás esélye meglehetősen nagy, azért a szkriptekkel történő helyreállítást javaslom.

A beszállítók és leányvállalatok változásai szintén nem mentődnek a rendszerben, ha az adatbázis nem elérhető. A probléma legegyszerűbb kezelése, ha szintén fájlba menti az ESB a változásokat addig, amíg az adatbázis újra elérhetővé válik.

Másik megoldás, hogy a beszállítók és a leányvállalatok nyitnak egy új szolgáltatást, mely egy dátumot vár paraméterként. A szolgáltatás meghívása után azokat a termékeket és adatait adja vissza a web szolgáltatás, amelyek az adott dátum óta megváltoztak. Ehhez tárolniuk kell a leányvállalatoknak és beszállítóknak, hogy az egyes termékek paraméterei mikor változtak meg utoljára. Amennyiben ez a dátum későbbi, mint a paraméterként kapott érték, akkor a termék adatai frissebbek, mint amit a koordinátor cég lát, így ezeket újra kell küldeni. A leányvállalatnak természetesen tárolnia kell a beérkezett rendelések időpontját ez a módszer esetén.

Harmadik módszerként az ESB egy erre a célra dedikált kivétellel jelezheti a beszállítóknak, hogy az adatbázis pillanatnyilag nem elérhető. A leányvállalatnak vagy beszállítóknak ekkor mentenie kell az üzenetet, és újraküldenie, amint az ESB jelzi egy szolgáltatás meghívásával, hogy az adatbázis újra elérhető. Célszerű ebben az esetben a leányvállalatnak vagy beszállítóknak lokálisan adatbázisba mentenie a változó termékek adatait, és amikor az ESB újra eléri saját adatbázisát, akkor annak nem minden üzenetet újra elküldeni, csak a jelenlegi állapotot leíró adatokat. A megvalósításban ezt a módszert választottam.

Amennyiben az ESB esik ki, akkor a részrendszerek között mindenfajta kommunikáció megszűnik, az ügyintézők nem tudnak belépni, adatbázist elérni, a beszállítók és leányvállalatok nem tudják a változásokat jelezni. Ebben az esetben megoldást jelent az adatbázis kiesését kezelő második és harmadik megoldás.

Az MQ kiesése esetén az ügyintézők nem tájékozódhatnak a változásokról, és nem tudnak újabb rendeléseket vagy lekérdezéseket küldeni az ESB felé. Inkonzisztens állapotba nem kerülhet így a rendszer, viszont az ügyintézők elvesztik a kapcsolatot a rendszerrel. Az MQ helyreállítása után a munkát problémamentesen tudják folytatni.

Ha a beszállítók nem elérhetőek, akkor újabb rendeléseket nem lehet leadni, illetve a változásokról sem tájékozódik a rendszer. Ebben az esetben az ESB-nek tájékoztatnia kell az ügyintézőket a beszállítók elérhetetlenségéről. A beszállítók rendszere, ha újra él, akkor az addig bekövetkezett változásokról tájékoztatnia kell az

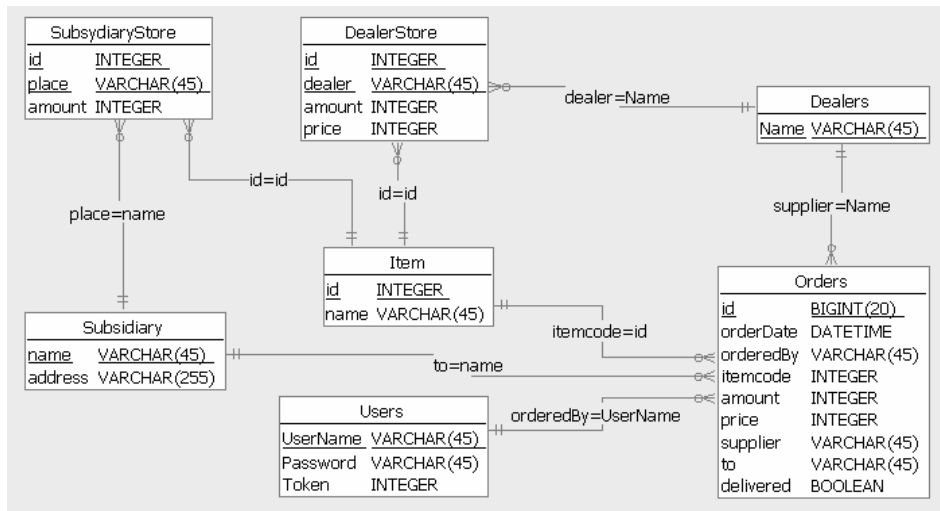
ESB-t. A rendszer a specifikáció szerint újrapróbálkozik a beszállítók szolgáltatásainak újraküldésével sikertelen hívás esetén. Ha az időtúllépés (timeout) értéke alacsony, és ezt hálózati terheltség miatt túllépi a hívás, akkor újraküldi azt az ESB. Ez viszont a rendelés kétszeres feladását jelenti, ha a beszállító rendszere működik, mivel a hívás nem idempotens. Ezt az anomáliát úgy lehet kiküszöbölni, hogy az ESB a hívással együtt küld egy generált azonosítót, melyet újraküldés esetén nem változtat meg. Ha a beszállító rendelést kap ugyanazzal az azonosítóval, akkor detektálni tudja, hogy ez egy rendelés újraküldése, és csak egyszer kell leszállítania.

3 Megvalósítás

A fejezet tartalmazza a megvalósított rendszer leírását az adatbázis által használt adattábláival kezdve az ESB-ben futó mediációs folyamatokon és beállításokon keresztül az MQ infrastruktúrán át a kliensoldali kódig. A fejezet végén találhatóak a rendszer továbbfejlesztési lehetőségei.

3.1 Adattáblák bemutatása

A rendszer egyes adatok mentésére adatbázist használ. Az alfejezet célja az adatbázis szerkezetének bemutatása. Az adattáblák és a közöttük levő kényszerek a 21. ábrán tekinthetők meg. A `Dealers` táblában található a beszállítók neve, a `DealerStore` táblában pedig a készletük. A `DealerStore` táblára láthatóan két kényszer vonatkozik: csak olyan beszállítónak szerepelhet benne bejegyzése, aki szerepel a `Dealers` táblában, és csak olyan termék, melynek azonosítója megtalálható az `Item` táblában. Az `Item` táblában a termékek leírói szerepelnek, azaz az azonosítója és a neve. Az adatbázis szerkezetéből látható, hogy a beszállítók egységes kódrendszert használnak, azaz megegyező azonosítóval ugyanazt a terméket jelölik, és az elnevezésében sem tesznek különbséget. A leányvállalatok neve és címe a `Subsidiary` táblában található, míg készletük a `SubsidiaryStore`-ban. Az előzőekkel anagalog módon kényszerek biztosítják, hogy a leányvállalat készletét leíró táblában csak létező leányvállalatnevek és termékek fordulhassanak elő. A `Users` tábla az ügyintézők felhasználónevét, jelszavát és azonosítótokenét tárolja. A rendelések táblában a beszállító által adott rendelésazonosító található kulcsként, és a tábla idegen kulcsok segítségével hivatkozik a rendelt termékre, a megrendelőre, a beszállítóra és a szállítási címre.



21. ábra Adattáblák bemutatása

3.2 MQ infrastruktúra bemutatása

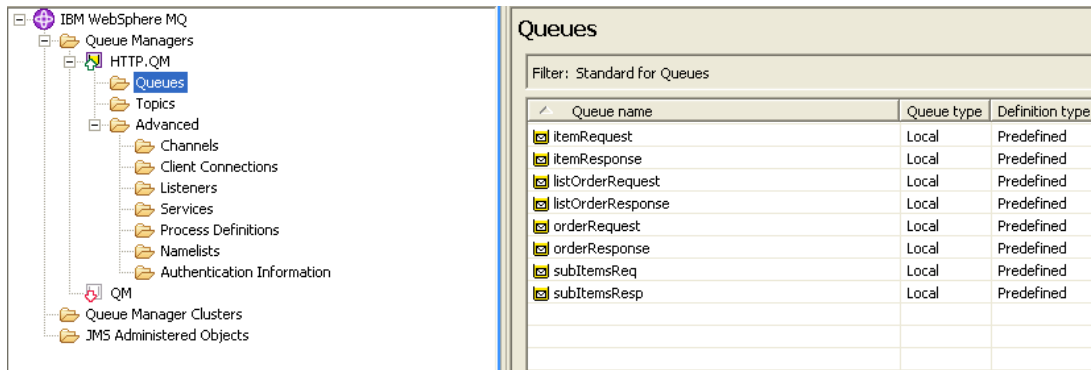
A fejezet célja az ESB és az ügyintézők által használt felület közötti kommunikációt biztosító MQ infrastruktúra és beállításainak bemutatása.

3.2.1 MQ üzenetsorok

Az ügyintézők kérései MQ üzenetsorokhoz továbbítódnak. Egy ügyintéző négyféle kérést küldhet, ezek a következők:

- leányvállalat készletének lekérdezése (subItems),
- rendelések listázása egy adott intervallumon belül (listOrders),
- adott kódú termék keresése (item),
- megadott mennyiségű termék rendelése megadott beszállítótól adott leányvállalathoz. (order)

Minden kéréshez tartozik egy külön üzenetsor, valamint a válaszhoz egy másik, így összesen nyolc üzenetsoron zajlik a kommunikáció. Az üzenetsorok a 22. ábrán láthatóak.



22. ábra A rendszerben használt üzenetsorok listája

Az üzenetsorokhoz minden kliens hozzáfér, ezért szükség van arra, hogy a válaszüzenetek címzettje meghatározható legyen. Erre azt a megoldást választottam, hogy az üzenet `CorrelationID` mezőjébe a belépett felhasználó nevét tettem, és az ESB-ben beállítottam, hogy a válaszüzenet `CorrelationID` mezője is ugyanazt az értéket tartalmazza.

3.2.2 Publish/subscribe üzenetküldés MQ segítségével

Az ügyintézők a leányvállalatok és beszállítók készletének változásairól publish/subscribe üzenetküldés segítségével értesülnek, melyhez az infrastruktúrát az MQ biztosítja. Ahhoz, hogy a Queue Manager publish/subscribe módon tudjon üzeneteket küldeni be kellett konfigurálni. Az IBM által biztosított `MQJMS_PSQ.mqsc` kódot le kellett futtatnom a Queue Manager-en, mely létrehozta az üzenetküldéshez a szükséges infrastruktúrát. A kód a függelékben megtalálható. A kód üzenetsorokat hoz létre. Az így létrehozott üzenetsorokra a 3.3.1 fejezetben bemutatott szervlet konfigurálásánál kellett hivatkoynom. Az MQ `SYSTEM.BROKER` szolgáltatásának elindítása után a feltételek adottak voltak a publish/subscribe üzenetküldéshez.

Az MQ-hoz tartozó Eclipse alapú grafikus felület – az MQ Explorer – alapértelmezés szerint nem támogatja a témák megjelenítését. A feladat teljesítéséhez így le kellett töltenem az `MS0Q`¹ nevű Eclipse plugint, s élesítése után láthattam MQ Explorer-en keresztül a létrejött témákat. A témák futási időben jönnek létre, amikor egy kliens csatlakozik az MQ-hoz, így a későbbiekben használt témákat nem kellett statikusan előre létrehoznom, további beállításokat nem kellett tennem.

¹ http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24013508&loc=en_US&cs=utf-8?en

3.3 ESB beállítások és folyamatok bemutatása

A fejezet az ESB-ben kialakított folyamatokat és beállításokat ismerteti. Ide tartozik:

- az MQ üzenetek AJAX segítségével történő manipulációja, mivel az ESB-ben futó szervlet segítségével történik
- az adatbázis elérés koncepciójának bemutatása
- az MQ üzenetek ESB üzleti objektummá alakítása, és fordítva
- a publish/subscribe üzenetküldés beállításainak bemutatása
- a tervezés fejezetben ismertetett folyamatok ESB megvalósításainak bemutatása

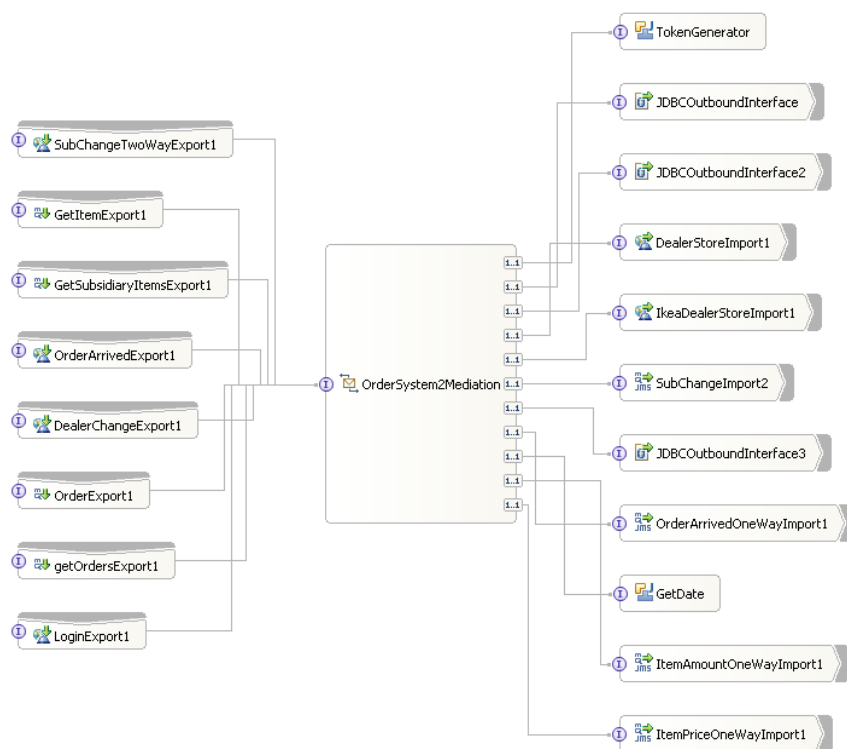
3.3.1 MQ elérése AJAX segítségével

AJAX segítségével közvetlenül nem lehet elérni a Message Queue által használt 6-os verzióját, viszont az IBM kiadott egy szervletet², melyet egy legalább J2EE 1.4 kompatibilis alkalmazáserverbe telepítve megoldható az üzenetküldés. A szervlet telepítése közben meg kell adni egy `context-root` változót, valamint létre kellett hoznom egy `Connection Factory` objektumot. A `Connection Factory` tartalmazta a szervlet MQ-hoz való kapcsolódásának adatait. Ezután a `http://hostname:serverport/context-root/msg/topic/topicname` címen elérhetőek az MQ témák és a `http://hostname:serverport/context-root/msg/queue/queueName` címen az MQ üzenetsorok. A hívásoknak három típusa lehet: `GET`, `PUT` és `DELETE`. `PUT` típusú hívásnál üzenetet lehet elhelyezni az üzenetsoron vagy témán, `GET` esetén olvasás hajtható végre és az üzenet megmarad az üzenetsoron, `DELETE` esetén pedig az üzenet olvasását követően törtődik az üzenet az üzenetsorról vagy témáról. Témákhoz nem lehetséges `GET` hívással csatlakozni. A hívások előtt a kérést fel lehet paraméterezni például azzal, hogy milyen azonosítójú üzenetet vár a hívás. Erről részletesebben a 3.4 fejezetben írok.

² <http://www-01.ibm.com/support/docview.wss?uid=swg24016142>

3.3.2 ESB összeszerelési diagram

Az ESB által összekötött rendszerkomponensek importjai és exportjai a 23. ábrán láthatóak. Közöttük az OrderSystem2Mediation komponens valósítja meg az összeköttetést. Az exportoknak az ikonjuk alapján láthatóan két típusa van: MQ export és Web Service export, azaz MQ és web szolgáltatás segítségével lehet adatokat vinni a rendszerbe. Tervezői döntés, hogy az ügyintézők MQ segítségével visznek adatokat a rendszerbe, a leányvállalatok és a beszállítók pedig web szolgáltatások segítségével. A SubChangeTwoWayExport1-en keresztül a leányvállalatok jelezhetik készleteik változását, a GetItemExport1-en keresztül keresnek rá az ügyintézők az egyes termékekre, a GetSubsidiaryItemsExport1-en keresztül a leányvállalatok készletei kérdezhetőek le, az OrderArrivedExport1-en keresztül jelzik a leányvállalatok a beérkezett rendeléseket, a DealerChangeExport1-en keresztül jelzik a beszállítók készletük változásait, az OrderExport1-en keresztül rendelnek az ügyintézők, a getOrdersExport1-en keresztül kéri le egy megadott intervallumon belüli rendeléseket, a LoginExport1-en keresztül pedig a belépés történik.



23. ábra Az ESB-ben futó rendszer képe

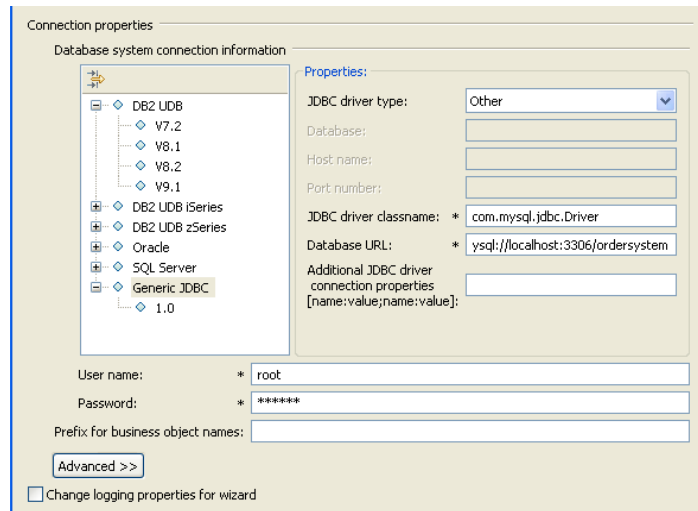
A `TokenGenerator` a felhasználók tokengeneráló logikáját tartalmazó Java komponens, a `JDBCOutboundInterface` változatai az adatbázis elérést biztosítják, a `DealerStoreImport`-ok segítségével érhetőek el a beszállítók rendelést biztosító web szolgáltatások, a `SubChangeImport2` segítségével történik a kliensek értesítése a leányvállalatok készletének változásairól, az `OrderArrivedOneWayImport1` segítségével történik a kliensek tájékoztatása a rendelések beérkezéséről, a `GetDate` a mai dátumot visszaadó Java komponens, az `ItemAmountOneWayImport1` segítségével történik az ügyintézők számára a beszállítók mennyiségi változásainak értesítése, az `ItemPriceOneWayImport1`-en keresztül pedig az árváltozások jelezhetőek.

3.3.3 Kapcsolat felépítése az adatbázissal

Az ESB-nek a 3.1 fejezetben leírt adatbázist el kell érnie. Ezt erőforrás adapteren (`Resource Adapter`) keresztül teszi meg. Az adapter elkészítését varázsló segíti. A `WebSphere Integration Developer` alapértelmezetten nem tartalmazza az általam használt `MySQL` adatbázishoz tartozó meghajtót, így azt külön le kellett töltenem³, és az adapter beállításainál hivatkoznom kellett a külső fájlra. A 24. ábrán látható módon be kellett konfigurálni a kapcsolat típusát (`JDBC`), a meghajtó osztályt (`com.mysql.jdbc.Driver`) valamint az adatbázis elérhetőségét és az autentikációhoz szükséges felhasználónév és jelszó párost. Az adatok megadása után következhetett az `SQL` lekérdezések kialakítása. A lekérdezéseket a fejlesztőkörnyezet validálta szintaktikailag, valamint a lekérdezések bemeneti paramétereinek és visszatérési értékeinek megfelelő üzleti objektumokat létrehozta.

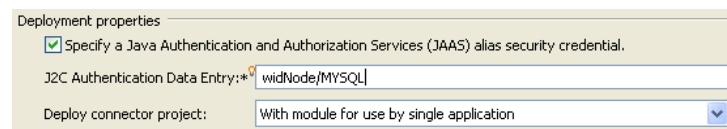
³

<http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.7.zip/from/ftp://ftp.crysys.hu/pub/mysql/>



24. ábra Erőforrás adapter létrehozása

Az erőforrás adapter létrehozásához az ESB-ben létre kellett hozni egy J2C autentikációs adat bejegyzést (J2C Authentication Data Entry), melyre a 25. ábrán látható módon kellett hivatkozni. Az ábrán látható, hogy a neve widNode/MYSQL. A bejegyzésben azt a felhasználónév és jelszó párost kellett megadni, amelyet az erőforrás adapter a feltelepített alkalmazásban használni fog.



25. ábra Autentikációs bejegyzés megadása

3.3.4 MQ üzenetek feldolgozása

Az ügyintézők által küldött üzenetek MQ üzenetsorokra kerülnek, melyeket az ESB feldolgoz, és a megfelelő folyamatra irányít. Ehhez az ESB-nek a 26. ábrán látható módon meg kell adni az üzenetsor elérhetőségét, a válasz üzenetsor nevét, az adatkötési konfigurációt és a függvényválasztó konfigurációt. Az adatkötési konfiguráció szerepe az, hogy az MQ üzeneteket ESB üzleti objektumokká transzformálja és fordítva. Az adatkötési konfigurációt Java nyelven kell megírni, és meg kell valósítania a `com.ibm.websphere.sca.mq.data.MQBodyDataBinding` interfészt. Az interfész által tartalmazott metódusok közül a két legfontosabb a `read(MQMD arg0, List arg1, MQDataInputStream arg2)` és a `write(MQMD arg0, List arg1, MQDataInputStream arg2)`. A `read` metódus akkor hívódik meg, ha üzenet érkezett a figyelt üzenetsorra, és azt üzleti objektummá kell alakítani, a `write` pedig akkor, ha az ESB a kérésre adott válasz üzleti objektumot vissza kell rakni egy

üzenetsorra MQ üzenet formátumban. A függelék tartalmazza a 26. ábrán hivatkozott `binders.OrderBinder` adatkötés `read` metódusát, mely az ügyintéző által küldött rendelésadatokat dolgozza fel. A kódból látható, hogy az MQ API meghatározott byteméretű részletek kiolvasására ad lehetőséget az üzenetből. Ha az üzenet méreténél nagyobb próbál a kód kiolvasni, akkor `EndOfFileException` kivétel dobódik. Ezt a problémát úgy hidaltam át, hogy kliensoldalon kitöltő karakterekkel akkorára növeltem, melynek beolvasására az MQ API lehetőséget nyújt, és az adatkötési konfigurációban a kitöltőkaraktereket leszedtem.

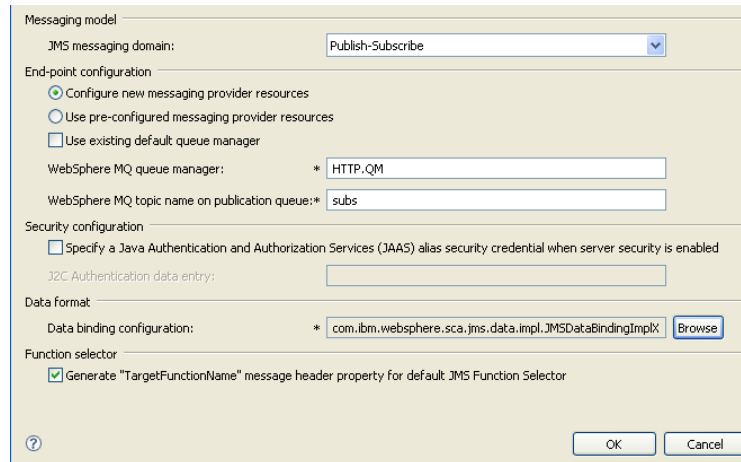
26. ábra MQ export beállítása

Az ábrán látható, hogy a válasz adatkötési konfigurációja `com.ibm.websphere.sca.mq.data.impl.MQDataBindingImplXML`. Ez az osztály egy előre legyártott adatkötési konfiguráció. Funkcióját tekintve az adatokat XML formátumban írja ki MQ-ra. Az ügyintézők számára küldött válasz generálásához ezt választottam, mivel kellően rugalmas struktúra, és DOM segítségével kényelmesen értelmezhető.

Az MQ üzenetsorok figyelése interfészekhez van kötve, viszont egy interfész számos metódussal rendelkezhet. Meg kell határozni, hogy ha egy üzenet érkezik az interfészhez rendelt üzenetsoron, akkor melyik metódusa hívodjon meg. Ezt a feladatot látja el a funkcióválasztó konfiguráció. Az általam választott funkcióválasztó a `com.ibm.websphere.sca.mq.selector.impl.Format` osztály, mely az MQ üzenet `Format` mezőjéből olvassa ki a meghívandó függvény nevét. Ez természetesen azzal jár, hogy az üzenet elküldése előtt a `Format` mező értékét be kell állítani.

3.3.5 Publish/Subscribe üzenetküldés

A rendszertervezés fejezetben eldöntöttem, hogy az ügyintézők tájékoztatása a változásokról publish/subscribe módon történik. Ezt az MQ támogatja, a Websphere Integration Developer pedig támogatást nyújtott az MQ témák eléréséhez. A 27. ábrán látható felületen az MQ queue manager és téma nevének valamint az adatkötési konfigurációnak a megadásával megvalósítható, hogy az ESB elérje az MQ témát és XML formátumban elhelyezze rajta az üzleti objektumokat.



The screenshot shows a configuration dialog box titled "Messaging model". It has several sections:

- Messaging model:** JMS messaging domain: Publish-Subscribe (dropdown menu).
- End-point configuration:**
 - Configure new messaging provider resources
 - Use pre-configured messaging provider resources
 - Use existing default queue manager
 - WebSphere MQ queue manager: * HTTP.QM (text field)
 - WebSphere MQ topic name on publication queue:* subs (text field)
- Security configuration:**
 - Specify a Java Authentication and Authorization Services (JAAS) alias security credential when server security is enabled
 - J2C Authentication data entry: (text field)
- Data Format:**
 - Data binding configuration: * com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplX (text field) with a "Browse" button.
- Function selector:**
 - Generate "TargetFunctionName" message header property for default JMS Function Selector

At the bottom right, there are "OK" and "Cancel" buttons.

27. ábra Publish/subscribe üzenetküldés beállítása

3.3.6 Mediációs folyamatok ismertetése

A fejezet a célja a 2.3 fejezetben ismertetett implementációfüggetlen forgatókönyvek ESB implementációjának ismertetése.

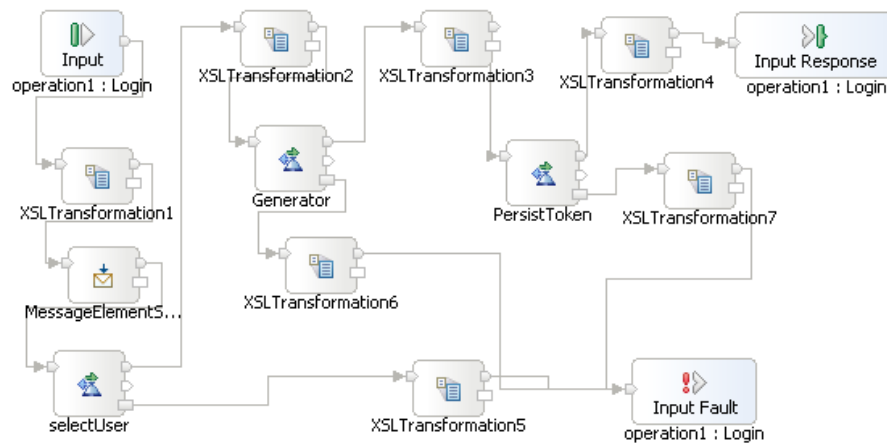
Ügyintéző bejelentkezése

Miután az ügyintéző webes felületen megadta felhasználónevét és jelszavát, egy szervlet hívódik meg, mely a 28. ábrán látható folyamatot hívja meg web szolgáltatásként.

Az `XSLTransformation1` és `MessageElementSetter1` elemek a `selectUser` szolgáltatásnak megfelelő üzleti objektumot állítják elő a bejövő adatokból, és kimentik a bemeneti adatokat tranzienstextusba. A `selectUser` szolgáltatás a felhasználónév és jelszó alapján keres rekordot a `users` táblában. Amennyiben nem talál megfelelő bejegyzést, vagy az adatbázis nem elérhető, akkor az üzenet az `XSLTransformation5` primitív felé folytatja útját, mely beállítja a visszaküldendő hibaüzenetet, és annak elküldésével a futás befejeződik. Ha szerepel a felhasználó az adatbázisban a megadott jelszóval, akkor a `Generator` szolgáltatás

generál neki egy azonosító token, a `PersistToken` szolgáltatás adatbázisba menti a felhasználóhoz tartozó frissen generált token, és az `XSLTransformation4` a tranzakció sikerességét jelentő üzenetet állít elő. Az `XSLTransformation6` és `XSLTransformation7` a `Generator` és az adatbázis hibájáról szóló hibaüzenetet generál.

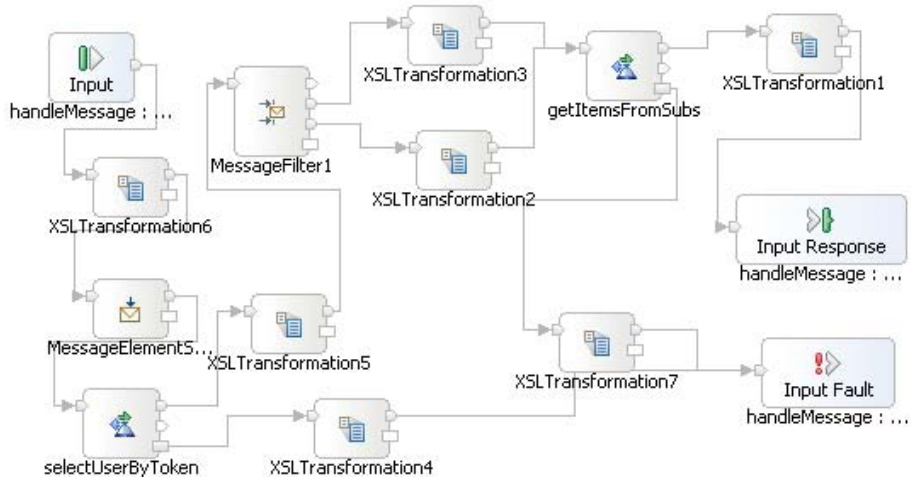
A szolgáltatás hibamentes lefutása után a szervlet átirányítja a felhasználót a 17. ábrán látható felületre, hiba esetén pedig a bejelentkezőoldalra, ahol megjeleníti a hibaüzenetet.



28. ábra A bejelentkezés megvalósítása

Leányvállalat készletének lekérdezése

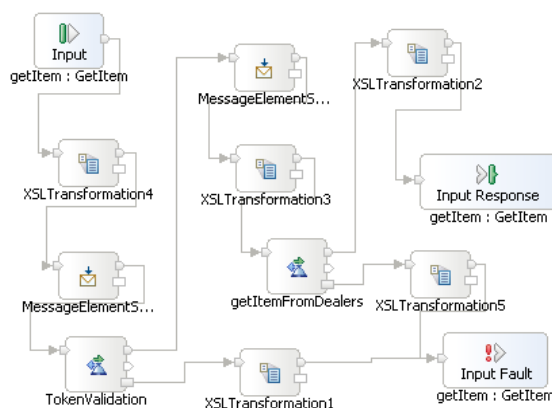
Az ügyintéző bejelentkezése után az oldal automatikusan lekérdezi a leányvállalatok készleteit. Ehhez elküldi AJAX segítségével az ügyintéző azonosítótokenjét és a leányvállalat azonosítóját, s ezekkel az adatokkal a 29. ábrán látható folyamat hívódik meg. Az első két primitív a `selectUserByToken` szolgáltatás interfészéhez alakítja a beérkező üzleti objektumot, és tranzienst kontextusba menti a bemeneti paramétereket. A `selectUserByToken` a küldött token alapján azonosítja a kérés küldőjét, majd a `MessageFilter1` kiválasztja, hogy melyik leányvállalat készletére kérdeztek rá, s az ennek megfelelő lekérdezést előállító XSL transzformációhoz irányítja az üzenetet. Amennyiben újabb beszállítóval bővül a rendszer, akkor újabb XSL transzformációval kell bővíteni a folyamatot. Végül meghívódik a `getItemsFromSubs`, mely visszaadja a leányvállalat készletét. A komponensek hibájához tartozó hibaüzenetet XSL transzformációk állítják elő itt is, és a `Fault` kimenetre irányítják.



29. ábra Leányvállalat készletének lekérézése

Termék lekérézése beszállítóktól

A folyamat a 30. ábrán tekinthető meg. Az leányvállalat készletének lekérézésével megegyezően tokenellenőrzéssel kezdődik a folyamat. Érvényes token esetén a MessageElementSetter1 és XSLTransformation3 primitívek előállítják a megfelelő lekérézést, mellyel a getItemFromDealers szolgáltatás hívódik meg. A szolgáltatás egy JDBC lekérézést valósít meg, mely az adott kódú terméknek a beszállítóknál elérhető mennyiségét és egységárát adja vissza.

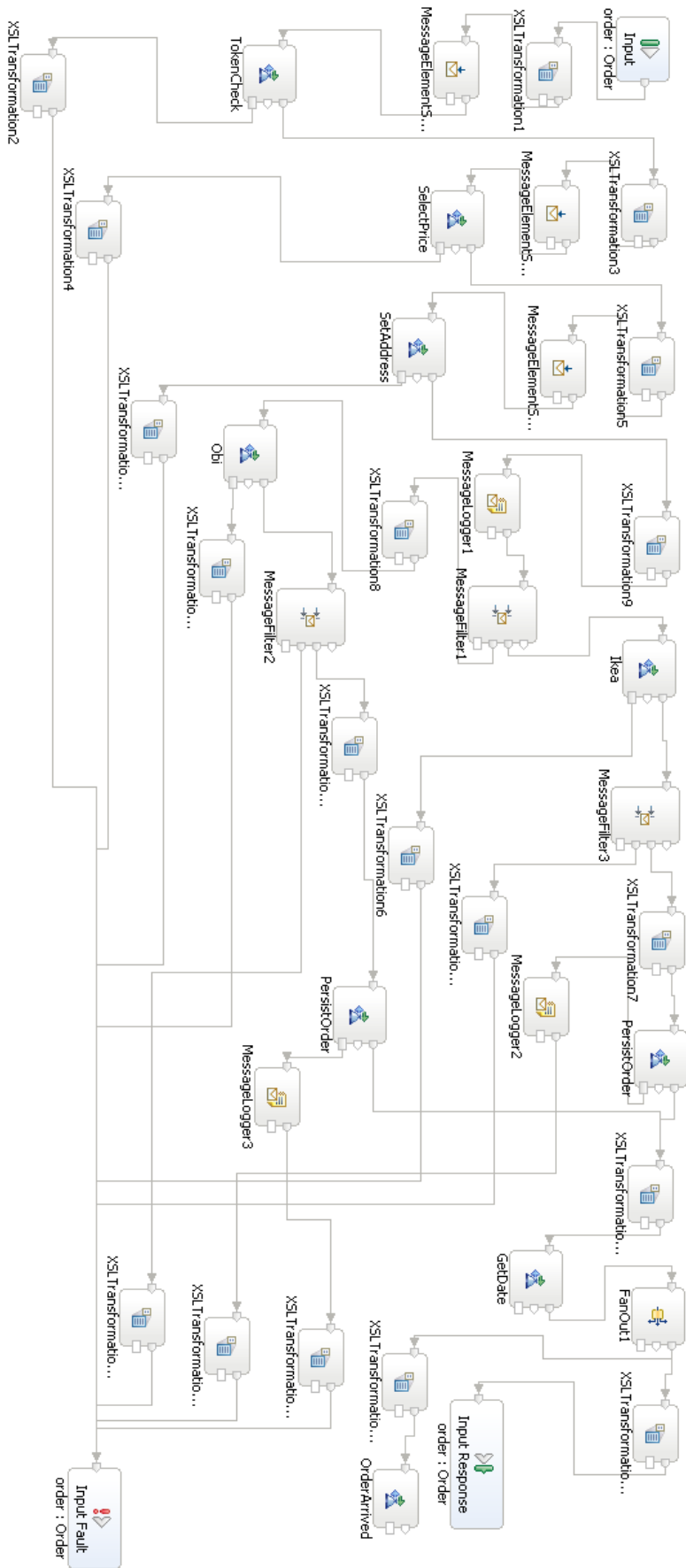


30. ábra Termék lekérézése beszállítóktól

Rendelés

A rendelés folyamata a 31. ábrán tekinthető meg. A folyamat az ügyintézőknél megszokott tokenellenőrzéssel kezdődik a TokenCheck szolgáltatás hívásával. Következő lépésben a termék árát, majd a leányvállalat címét kérdezi le a folyamat a SelectPrice és a SetAddress hívásokkal. A szolgáltatások hívása előtti XSL transzformációk és MessageElementSetter primitívek szerepe a bemeneti

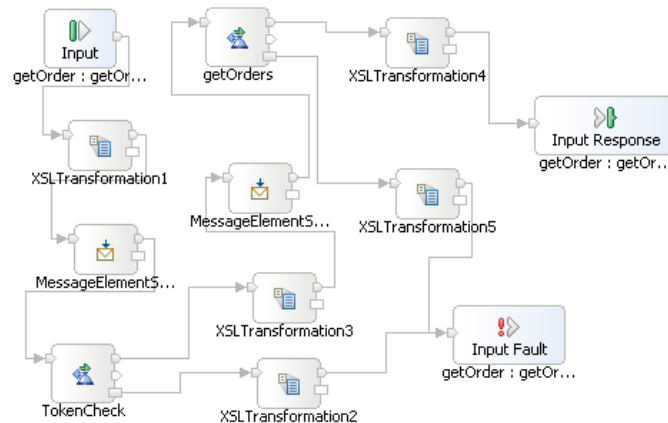
adatokból a szolgáltatások hívásához szükségesek kiválasztása, a szolgáltatások utáni XSL transzformációk pedig a lekérdezések eredményét mentik tranziens kontextusba későbbi használatra. A `MessageLogger1` segítségével megtörténik az üzenet logolása, majd a `MessageFilter1` a beszállító paraméter értékének megfelelően a megfelelő beszállító web szolgáltatásához irányítja az üzenetet. A beszállító web szolgáltatásainak meghívása után a `MessageFilter2` és `MessageFilter3` a válasz típusának függvényében – mely lehet `ERROR` vagy `SUCCESS` – irányítja az üzenetet. Amennyiben `ERROR` típusú a válasz, akkor az üzenet olyan ágon folytatja útját, melyen az elutasításhoz tartozó hibaüzenetet egy XSL transzformáció előállítja. Amennyiben elfogadta a beszállító a rendelést, akkor a rendelés adatait a `PersistOrder` szolgáltatás menti a visszakapott rendelésazonosítóval együtt. Amennyiben az adatbázis nem elérhető `MessageLogger` primitív segítségével lementi a folyamat a rendelés adatait, és XSL transzformációval előállítja a hibaüzenetet. Ha a rendelés adatait sikerült menteni a `PersistOrder` szolgáltatással, akkor a folyamat a `GetDate` szolgáltatással lekéri az aktuális dátumot, majd párhuzamosan előállítja a rendelés elfogadásához tartozó visszajelzést, és az `OrderArrived` szolgáltatás aktuális dátummal történő hívásával jelzi, hogy az aznapi rendelések halmaza megváltozott.



31. ábra Rendelés

Rendelések lekérdezése

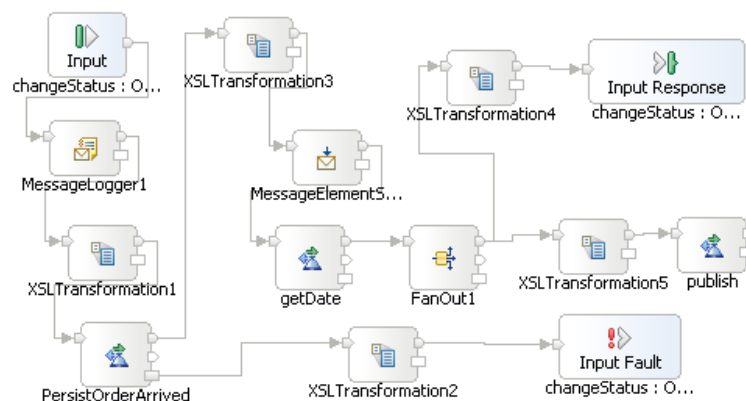
Az ügyintézőknek lehetőségük van az általuk megadott intervallumon kilistáztatni a rendeléseket. A folyamat megtekinthető a 32. ábrán. A szokásos tokenellenőrzés után a `getOrders` szolgáltatás az `orders` táblán lefuttatja a szükséges lekérdezést, majd eredményét XML formátumban visszaadja. Autentikációs és adatbáziselési probléma esetén hibaüzenetben tájékoztatja a hibáról a felhasználót.



32. ábra Rendelések lekérdezése

Rendelés beérkezése

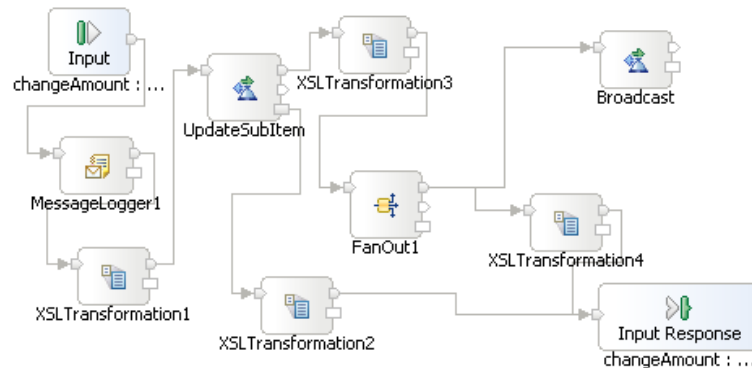
Ha egy rendelés megérkezik a leányvállalathoz, akkor a leányvállalat a nevének, a rendelés azonosítójának és a beszállító nevének megadásával meghív egy ESB-ben futó web szolgáltatást, mely az alábbi folyamatot indítja el. A folyamat az elején logolja a beérkezett üzenetet, a `PersistOrderArrived` az `orders` táblában beállítja a megadott azonosító rendelés státuszát kézbesítettre, majd lekéri a rendelés dátumát. A folyamat a végén párhuzamosan egyrészt MQ témán keresztül jelzi a rendelés dátumának elküldésével, hogy az aznapi rendelések között változás történt, másrészt visszajelez a hívónak hívása sikeres feldolgozásáról.



33. ábra Rendelés beérkezésének folyamata

Leányvállalat készletének változása

A leányvállalatok a 34. ábrán látható folyamat segítségével jelentik készletük változását azonosítójuk, a termék kódjának és mennyiségének megadásával. A hívást a rendszer logolja, az UpdateSubItem szolgáltatás segítségével menti az adatbázisba, majd sikeres mentés esetén párhuzamosan visszajelez a hívó félnek a változás sikeres feldolgozásáról és a Broadcast szolgáltatás segítségével MQ témán keresztül jelzi a belépett ügyintézők számára a változást.



34. ábra Leányvállalat készletváltozásának jelentése

Beszállítók termékeinek változásai

A beszállítók termékeinek egységáráról és mennyiségi változásáról szóló tájékoztatás az előző folyamattal analóg módon történik. A beszállítók hívását logolja az ESB, adatbázisba menti a változást, majd MQ témán keresztül jelzi a változást a belépett ügyintézők számára.

3.4 A kliensoldali kód ismertetése

A fejezet célja az ügyintézők által beállított adatok küldéséért és feldolgozásáért felelős kód ismertetése. A kódok két csoportba oszthatóak:

- témát figyelő kódok
- kérést üzenetsorra küldő és válaszra váró kódok

Az egy csoportba tartozó kódok működési koncepciója hasonló, általában csak a válasz feldolgozásában különböznek, így a két típusú kódból egy-egy ismertetése megfelelő rálátást ad működésükre. Ez alapján a fejezetben két típusból egy-egy ismertetését írom le. A kódok megtalálhatóak egészben a függelék részben.

3.4.1 Témát figyelő kódok

A témát figyelő kódok MQ témákról olvasnak adatokat, amíg az ügyintéző be van jelentkezve. Feladatuk a rendszerben történő változások érzékelése, és szükség esetén a tartalom frissítése vagy a frissítés kezdeményezése. A témát figyelő kódok a belépéskor automatikusan meghívódnak. Az ismertetésre kiválasztott kód a rendelések változásait figyelő kód.

Első lépésben a kód létrehozza az AJAX kéréshez szükséges XMLHttpRequest objektumot, vagy tájékoztat arról, hogy az nem hozható létre.

```
if (window.XMLHttpRequest) {
    req_order_watch = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req_order_watch = new ActiveXObject("Microsoft.XMLHTTP");
} else {
    alert("A böngészője nem támogatja az AJAX-ot...");
}
```

A következő lépésben a kód meghatározza a kérés címét. Ez a cím az általam bekonfigurált szervlet elérhetősége. A címből látható, hogy az orders témához nyújt hozzáférést a megadott cím.

```
var order_watch_addr = "http://localhost:9082/esbmq/msg/topic/orders";
```

Ezt a kérés feldolgozásának meghatározása követi. A kódból láthatóan ez névtelen függvénnyel történik. Amennyiben a kérésre megérkezett a válasz (readyState==4), és hiba nélkül érkezett meg a válasz (status==200), akkor kezdeményezi a DOM segítségével történő értelmezését.

```
req_order_watch.onreadystatechange = function() {
    if(req_order_watch.readyState==4)
    {
        if(req_order_watch.status==200){
            text=req_order_watch.responseText;
            try //Internet Explorer
            {
                xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
                xmlDoc.async="false";
                xmlDoc.loadXML(text);
            }
            catch(e)
            {
                try //Firefox, Mozilla, Opera, etc.
                {
                    parser=new DOMParser();
                    xmlDoc=parser.parseFromString(text, "text/xml");
                }
                catch(e)
                {
                    alert(e.message);
                    return;
                }
            }
        }
    }
}
```

A kód következő része feldolgozza a beérkezett üzenetet. Ebben az esetben a kapott üzenet `date` mezőjében található szöveg tartalmazza a dátumot, melykor egy rendelés megváltozott vagy keletkezett.

```
var changedDate =
xmlDoc.getElementsByTagName("date")[0].childNodes[0].nodeValue;

var changedYear = changedDate.substring(0,4);
var changedMonth = changedDate.substring(5,7);
var changedDay = changedDate.substring(8,10);

changedDate = new Date();
changedDate.setFullYear(changedYear,changedMonth-1,changedDay);
changedDate.setHours(0);
changedDate.setMinutes(0);
changedDate.setSeconds(0);
changedDate.setMilliseconds(0);
```

A következő lépésben el kell döntenie a kódnak, hogy mit kezdjen a kapott adattal. Jelen esetben azt kell eldöntenie, hogy frissítenie kell-e az ügyintéző által figyelt rendeléseket tartalmazó nézetet. Az oldal a `dateFrom` és `dateTo` JavaScript változóiban tárolja az ügyintéző által küldött rendeléslekérdezés intervallumának határait. A lenti kód ellenőrzi, hogy a kapott dátum a megjelenített intervallumba esik-e és szükség esetén a `getOrders()` hívással frissíti a táblázatot. Végül újra csatlakozik a témához.

```
if ((dateFrom!=null) && (dateTo!=null) && (changedDate>=dateFrom) &&
(changedDate<=dateTo))
{
    getOrders();
} else {
    //do nothing, no need to refresh table
}
watchOrdersTopic();
```

A hibakezelés az alábbi kódban valósul meg. Ha 504-es hibakóddal tér vissza az `XMLHttpRequest` objektum hívása, akkor a megadott timeout értéken belül nem érkezett üzenet a témára. Ekkor egyszerűen újra csatlakoznia kell a témához. Amennyiben nem 504-es a hibakód, akkor a hívás közben hiba lépett fel. Ekkor tájékoztatja a felhasználót a hiba okáról és kódjáról, aminek alapján a rendszer üzemeltetője intézkedhet.

```
} else {
    if (req_order_watch.status===504)
        //timeout
        watchOrdersTopic();
    } else {
        alert(req_order_watch.status+req_order_watch.responseText); } } }
```

Végül elküldi az AJAX kérést a megadott címre paraméter nélkül. Láthatóan az időtúllépés értékének 35 másodperc van beállítva.

```
req_order_watch.open("DELETE", order_watch_addr,true);
req_order_watch.setRequestHeader("Content-type", "text/plain");
req_order_watch.setRequestHeader('x-msg-wait', "35000");
req_order_watch.send(null);
```

3.4.2 Kérést üzenetsorra küldő és válaszra váró kódok

Az ebben az alfejezetben bemutatott kódokra jellemző, hogy kérés-válasz típusúak, általában egy gomb megnyomására aktiválódnak, és a hozzájuk tartozó válasz csak a kérés küldőjére vonatkozik. A bemutatásra kiválasztott kód a rendeléseket lekérdező kód.

Első lépésben két XMLHttpRequest objektumot hoznak létre a kérés elküldésére és a válasz üzenetsorról történő levételére, majd meghatározzák, hogy mely üzenetsorra helyezik a kérésüzenetet és a választ melyiken várják. Láthatóan jelen esetben az objektumok req és resp nevűek, és az üzenetsorok a listOrderRequest és listOrderResponse nevűek.

```
if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
    resp = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
    resp = new ActiveXObject("Microsoft.XMLHTTP");
} else {
    alert("A böngészője nem támogatja az AJAX-ot...");
}

var requestAddress = "http://localhost:9082/esbmq/msg/queue/listOrderRequest";
var responseAddress = "http://localhost:9082/esbmq/msg/queue/listOrderResponse";
```

Következő lépésben történik a kérés státuszának változásaihoz rendelt függvény leírása. Ha a kérés küldése eljutott a 2-es állapotba, azaz a kérés el van küldve, akkor a kód kezdeményezi a válaszüzenetsorhoz való csatlakozást. Érdeemes megfigyelni, hogy fejléc információként be van állítva a felhasználó felhasználóneve. Ezzel a kód azt határozza meg, hogy azt az üzenetet akarja levenni az üzenetsorról, amelynek

fejléc részében az ő felhasználóneve van, azaz azt, amelyik az ő kérésére érkezett válaszként.

```
req.onreadystatechange = function()
{
    if(req.readyState==2)
    {
        resp.open("DELETE",responseAddress,true);
        resp.setRequestHeader("x-msg-wait", "25000");
        resp.setRequestHeader("x-msg-correlId", username);
        resp.send(null);
    }
}
```

A kérés elküldésekor általános koncepció, hogy a kérést indító gombot a kód letiltja és szövegét megváltoztatja, hogy az ügyintéző tudja, hogy kérése feldolgozás alatt áll. A válasz megérkezésekor a lenti kód segítségével ezeket visszaállítja a kód, jelezve ezzel, hogy a válasz megérkezett a kérésre.

```
resp.onreadystatechange = function()
{
    if(resp.readyState==4)
    {
        if (resp.status==200){
            document.getElementById("OrderSearchButton").disabled=false;
            document.getElementById("OrderSearchButton").value="Keresés";
        }
    }
}
```

A válasz értelmezése az előző fejezethez hasonlóan DOM értelmezővel történik, majd attól eltérően az oldal tartalmának megváltoztatásával folytatódik. Jelen esetben a rendelések táblázat tartalmát törli a kód, majd minden rendeléstételnél egy új sort ad a táblázathoz, melynek adatait az aktuális rendeléstétel adataival tölti ki.

```
while (document.getElementById("orderResults").rows[1]!=undefined)
{
    document.getElementById("orderResults").deleteRow(1);
}

for(i=0;i<xmlDoc.getElementsByTagName("itemcode").length;i++){
var newrow = document.getElementById("orderResults").insertRow(i+1);
var datumszlop=newrow.insertCell(0);
var rendlooszlop=newrow.insertCell(1);
var kodoszlop=newrow.insertCell(2);
var nevoszlop=newrow.insertCell(3);
var mennyisegoszlop=newrow.insertCell(4);
var aroszlop=newrow.insertCell(5);
var beszallitooszlop=newrow.insertCell(6);
var cimoszlop=newrow.insertCell(7);
var szallitvaoszlop=newrow.insertCell(8);

datumszlop.innerHTML=xmlDoc.getElementsByTagName("orderdate")[i].childNodes[0].nodeValue.substring(0,10);
}
```



```
req.setRequestHeader("Content-type", "text/plain");
req.setRequestHeader("x-msg-format", "getOrder");
req.setRequestHeader("x-msg-correlId", username);
req.send(token+"|"+dateFromWithPadding+"|"+dateToWithPadding);
document.getElementById("OrderSearchButton").disabled=true;
document.getElementById("OrderSearchButton").value="Keresés folyamatban";
```

3.5 Továbbfejlesztési lehetőségek

Annak ellenére, hogy a rendszer számos funkciót támogat, bevezetéséhez további funkciók megvalósítására és a beállítások bekonfigurálására van szükség.

A rendszernek szüksége van egy kiemelt jogokkal rendelkező admin szerepkörre, akinek feladata a felhasználói fiókok menedzselése. Ehhez létre kell hozni az ezt támogató szolgáltatásokat és adatbázishátteret.

A rendszer jelenlegi megvalósításából hiányoznak a beszállítók web szolgáltatásai, azokat a rendelési igényeket véletlenszerűen elfogadó vagy elutasító csonkok helyettesítik. Ezt a leegyszerűsített modult tovább kell fejleszteni, szem előtt tartva például azt, hogy csak autentikáció után lehessen rendelni a web szolgáltatásukon keresztül, illetve kezelnie kell a konkurens kéréseket.

A rendszer jelenleg nem támogatja szolgáltatáson keresztül új termék felvételét.

A rendszer web szolgáltatásai autentikáció nélkül hívhatóak, és egyszerű szöveggként utaznak a hálózaton. Ennek a biztonsági résznek a kiküszöbölésére a WS-Security alkalmas szabvány.

A leányvállalatok és beszállítók számára szükség van egy-egy kliensre, melyet autentikáció után tudnak használni, és segítségével könnyen tudják jelezni a rendelések beérkezését illetve készletváltozásukat.

A rendszerben jelenleg biztonsági rés, hogy az MQ-ra bárki autentikáció nélkül üzenetet tud elhelyezni vagy levenni róla. Ennek kiküszöbölésére korlátozni kell a hozzáférést.

4 Összefoglalás

A diplomatervezés keretében megismerkedtem integrációs mintákkal, integrációt megkönnyítő technológiákkal és kliensoldali technikákkal. Az szerzett ismereket felhasználva megterveztem és létrehoztam egy e-Business rendszert IBM WebSphere termékek segítségével. A rendszer tervezése során meghatároztam a rendszer felhasználóinak szerepköreit, az előforduló forgatókönyveket, figyelembevéve a minimális erőforráshasználatot és kiválasztottam a rendszer megvalósításához szükséges komponenseket.

A megvalósítás során az alábbi feladatokat végeztem el:

- Kialakítottam a fejlesztő és futatókörnyezetet egy virtuális gépen, így IBM WebSphere Integration Developer-t, IBM WebSphere Message Queue-t, IBM WebSphere Enterprise Service Bus-t és MySQL adatbázist telepítettem.
- Az adatbázis struktúráját a tervek alapján megvalósítottam.
- Message Queue-ban létrehoztam a szükséges üzenetsorokat és elvégeztem a publish/subscribe üzenetküldéshez szükséges beállításokat.
- ESB-ben létrehoztam a tervezés fejezetben dokumentált folyamatokat, beállítottam, hogy az ESB tudjon adatbázishoz kapcsolódni, MQ témákra üzenetet küldeni, MQ üzenetsorokról üzenetet olvasni és azokra írni.
- ESB-be telepítettem egy szervletet, melyen keresztül AJAX segítségével elérhetőek az MQ üzenetsorok és témák.
- Létrehoztam AJAX és JavaScript alapú kliensoldali kódot, mely az üzenetek aszinkron küldését, fogadását és feldolgozását végzi.

Irodalomjegyzék

- [1] www.storrconsulting.com/images/eai-spaghetti.jpg
- [2] Paula Callister: Connecting to WebSphere Enterprise Service Bus (and WPS) with Adapters, IBM
- [3] Balogh Péter: Folyamatmegoldások SOA környezetben - SOA alapok, Budapesti Műszaki és Gazdaságtudományi Egyetem, <http://www.aut.bme.hu/Portal/Default/DocDownload.aspx?DocId=bbc56aea-bf8d-4e1d-8467-58677203aa63&CultureId=16df90ec-fcf2-466d-8f3e-8c4057561621>
- [4] <http://hu.wikipedia.org/wiki/XML>
- [5] http://nuwanbando.com/wp-content/uploads/2007/09/pict_uddi_1.gif
- [6] Mark Colan: SOA and Web Services for Architects and Developers, IBM, <ftp://www6.software.ibm.com/software/developer/library/mcolan/040816%20Colan,%20SOA%20and%20Web%20Services%20for%20Architects%20and%20Developers.pdf>
- [7] Balogh Péter: Folyamatmegoldások SOA környezetben - Alkalmazás Integráció, ESB, Budapesti Műszaki és Gazdaságtudományi Egyetem, <http://www.aut.bme.hu/Portal/Default/DocDownload.aspx?DocId=0a755aaf-4c37-42aa-81ef-d86c2f658cf0&CultureId=16df90ec-fcf2-466d-8f3e-8c4057561621>
- [8] IBM Software Group: Introducing WebSphere ESB
- [9] IBM Software Group: WESB Mediation Primitives

Függelék

MQ publish/subscribe üzenetküldés üzenetsorait létrehozó kód:

```
*****/
* IBM Websphere MQ Support for Java Message Service */
* Sample MQSC source defining JMS Publish/Subscribe queues. */
* Installation Verification Test - Setup script */
* */
* Licensed Materials - Property of IBM */
* */
* 5724-H72 5655-L82 5724-L26 */
* */
* (c) Copyright IBM Corp. 1999, 2005. All Rights Reserved. */
* */
* US Government Users Restricted Rights - Use, duplication or */
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
*****/
*****/
* */
* JMS Publish/Subscribe Administration Queue */
* */
*****/
** Create a local queue
  DEFINE QLOCAL('SYSTEM.JMS.ADMIN.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - admin queue') +
* Persistent messages OK
  DEFPSIST(YES) +
* Non-Shareable
  NOSHARE
*****/
* */
* JMS Publish/Subscribe Subscriber Status Queue */
* */
*****/
** Create a local queue
  DEFINE QLOCAL('SYSTEM.JMS.PS.STATUS.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - PS status queue') +
* Persistent messages OK
  DEFPSIST(YES) +
* Shareable
  SHARE DEFSOPT(SHARED)
*****/
* */
* JMS Publish/Subscribe Report Queue */
* */
*****/
** Create a local queue
  DEFINE QLOCAL('SYSTEM.JMS.REPORT.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - Report queue') +
* Persistent messages OK
  DEFPSIST(YES) +
* Shareable
  SHARE DEFSOPT(SHARED)
*****/
* */
```

```

* JMS Publish/Subscribe Subscribers Model Queue          */
*                                                         */
* Create model queue used by subscribers to create a permanent */
* queue for subscriptions                                */
*                                                         */
*****/
* General reply queue                                  */
DEFINE QMODEL('SYSTEM.JMS.MODEL.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - Model queue') +
* Queue Definition Type
  DEFTYPE(PERMDYN) +
* Shareable
  SHARE DEFSOPT(SHARED)
*****/
*                                                         */
* JMS Publish/Subscribe Default Non-Durable Shared Queue */
*                                                         */
* Create local queue used as the default shared queue by */
* non-durable subscribers                                */
*                                                         */
*****/
** Create a local queue
DEFINE QLOCAL('SYSTEM.JMS.ND.SUBSCRIBER.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - PS ND shared queue') +
* Persistent messages OK
  DEFPSIST(YES) +
* Shareable
  SHARE DEFSOPT(SHARED) +
* Maximum queue depth
  MAXDEPTH(100000)
*****/
*                                                         */
* JMS Publish/Subscribe Default Non-Durable Shared Queue for */
* ConnectionConsumer functionality                       */
*                                                         */
* Create local queue used as the default shared queue by */
* non-durable connection consumers                       */
*                                                         */
*****/
** Create a local queue
DEFINE QLOCAL('SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - PS ND CC shared q') +
* Persistent messages OK
  DEFPSIST(YES) +
* Shareable
  SHARE DEFSOPT(SHARED) +
* Maximum queue depth
  MAXDEPTH(100000)
*****/
*                                                         */
* JMS Publish/Subscribe Default Durable Shared Queue     */
*                                                         */
* Create local queue used as the default shared queue by durable */
* subscribers                                             */
*                                                         */
*****/
** Create a local queue
DEFINE QLOCAL('SYSTEM.JMS.D.SUBSCRIBER.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - PS D shared queue') +
* Persistent messages OK

```

```

    DEFPSIST(YES) +
* Shareable
  SHARE DEFSOPT(SHARED) +
* Maximum queue depth
  MAXDEPTH(100000)
*****/

*
* JMS Publish/Subscribe Default Durable Shared Queue for
* ConnectionConsumer functionality
*
* Create local queue used as the default shared queue by durable
* connection consumers
*

*****/

** Create a local queue
  DEFINE QLOCAL('SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE') REPLACE +
  DESCR('Websphere MQ - JMS Classes - PS D CC shared q') +
* Persistent messages OK
  DEFPSIST(YES) +
* Shareable
  SHARE DEFSOPT(SHARED) +
* Maximum queue depth
  MAXDEPTH(100000)

```

MQ üzenet átalakítása ESB üzleti objektummá:

```

public void read(MQMD arg0, List arg1, MQDataInputStream arg2)
    throws IOException {

    dataObject = DataFactory.INSTANCE.create("http://OrderSystem2","OrderDetails");

    int token;
    int itemId;
    String from;
    int amount;
    String to;

    try{
        //read 128 bytes from MQ
        String mqString = arg2.readMQCHAR128();
        //split the mqstring with "|" symbol
        String[] stringArray = mqString.split("\\|");
        //parse string
        token = new Integer(stringArray[0].replace(" ", ""));
        itemId = new Integer(stringArray[1].replace(" ", ""));
        from = stringArray[2].replace(" ", "");
        amount = new Integer(stringArray[3].replace(" ", ""));
        to = stringArray[4].replace(" ", "");
    } catch (Exception e) {
        //read or parse error
        e.printStackTrace();
        return;
    }
}

```

```

// map sub names from code to name
if (to.equals("00"))
{
    to="Budapesti kirendeltség";
} else if(to.equals("01"))
{
    to="Kecskeméti kirendeltség";
}

//set dataobject fields
dataObject.setInt(0, token);
dataObject.setInt(1, itemId);
dataObject.setString(2, from);
dataObject.setInt(3, amount);
dataObject.setString(4, to);
}

```

Kliensoldali publish/subscribe alapú művelet kód példa

```

function watchOrdersTopic()
{
if (window.XMLHttpRequest) {
    req_order_watch = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    req_order_watch = new ActiveXObject("Microsoft.XMLHTTP");
} else {
    alert("A böngészője nem támogatja az AJAX-ot...");
}

var requestAddress3 = "http://localhost:9082/esbmq/msg/topic/orders";

req_order_watch.onreadystatechange = function(){
    if(req_order_watch.readyState==4)
        {
            if(req_order_watch.status==200){

                text=req_order_watch.responseText;

                try //Internet Explorer
                {
                    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
                    xmlDoc.async="false";
                    xmlDoc.loadXML(text);
                }
                catch(e)
                {
                    try //Firefox, Mozilla, Opera, etc.
                    {
                        parser=new DOMParser();
                        xmlDoc=parser.parseFromString(text,"text/xml");
                    }
                    catch(e)
                    {
                        alert(e.message);
                    }
                }
            }
        }
    return;
}

```

```

    }
    }

    var changedDate =
xmlDoc.getElementsByTagName("field1")[0].childNodes[0].nodeValue;

    var changedYear = changedDate.substring(0,4);
    var changedMonth = changedDate.substring(5,7);
    var changedDay = changedDate.substring(8,10);

    changedDate = new Date();
    changedDate.setFullYear(changedYear,changedMonth-1,changedDay);
    changedDate.setHours(0);
    changedDate.setMinutes(0);
    changedDate.setSeconds(0);
    changedDate.setMilliseconds(0);

    if ((dateFrom!=null) && (dateTo!=null) && (changedDate>=dateFrom) &&
(changedDate<=dateTo))
    {
        //must refresh content
        getOrders();
    } else {
        //do nothing, no need to refresh table
    }

    watchOrdersTopic();
    }
    else {
        if (req_order_watch.status==504)
        {
            //timeout
            watchOrdersTopic();
        }
        else {
            // error happened

            alert(req_order_watch.status+req_order_watch.responseText);
        }
    }
}

req_order_watch.open("DELETE",requestAddress3,true);
req_order_watch.setRequestHeader("Content-type", "text/plain");
req_order_watch.setRequestHeader('x-msg-wait', "35000");
req_order_watch.send(null);
}

```

Kliensoldali kérés válasz alapú művelet kód példa

```

function getOrders()
{
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
        resp = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
        resp = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

```

```

    } else {
    alert("A böngészője nem támogatja az AJAX-ot...");
    }

var requestAddress = "http://localhost:9082/esbmq/msg/queue/listOrderRequest";
var responseAddress = "http://localhost:9082/esbmq/msg/queue/listOrderResponse";

req.onreadystatechange = function()
{
    if(req.readyState==2)
    {
        resp.open("DELETE",responseAddress,true);
        resp.setRequestHeader('x-msg-wait', "25000");
        resp.setRequestHeader("x-msg-correlId", username);
        resp.send(null);
        resp.onreadystatechange = function()
        {
            if(resp.readyState==4)
            {
                document.getElementById("OrderSearchButton").disabled=false;
                document.getElementById("OrderSearchButton").value="Keresés";

                text=resp.responseText;

                try //Internet Explorer
                {
                    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
                    xmlDoc.async="false";
                    xmlDoc.loadXML(text);
                }
                catch(e)
                {
                    try //Firefox, Mozilla, Opera, etc.
                    {
                        parser=new DOMParser();
                        xmlDoc=parser.parseFromString(text,"text/xml");
                    }
                    catch(e)
                    {
                        alert(e.message);
                        return;
                    }
                }
            }
        }

        while (document.getElementById("orderResults").rows[1]!=undefined)
        {
            document.getElementById("orderResults").deleteRow(1);
        }

        for(i=0;i<xmlDoc.getElementsByTagName("itemcode").length;i++){
            var newrow = document.getElementById("orderResults").insertRow(i+1);
            var datumszlop=newrow.insertCell(0);
            var rendelooszlop=newrow.insertCell(1);
            var kodoszlop=newrow.insertCell(2);
            var nevoszlop=newrow.insertCell(3);
            var mennyisegoszlop=newrow.insertCell(4);
            var aroszlop=newrow.insertCell(5);
            var beszallitooszlop=newrow.insertCell(6);
            var cimoszlop=newrow.insertCell(7);
            var szallitvaoszlop=newrow.insertCell(8);
        }
    }
}

```

```

        datumoszlop.innerHTML+xmlDoc.getElementsByTagName("orderdate")[i].childNodes[0].nodeValue.substring(0,10);

        rendelooszlop.innerHTML+xmlDoc.getElementsByTagName("orderedby")[i].childNodes[0].nodeValue;
        kodoszlop.innerHTML+xmlDoc.getElementsByTagName("itemcode")[i].childNodes[0].nodeValue;

        nevoszlop.innerHTML+xmlDoc.getElementsByTagName("name")[i].childNodes[0].nodeValue;

        mennyisegoszlop.innerHTML+xmlDoc.getElementsByTagName("amount")[i].childNodes[0].nodeValue;

        aroszlop.innerHTML+xmlDoc.getElementsByTagName("price")[i].childNodes[0].nodeValue;

        beszallitooszlop.innerHTML+xmlDoc.getElementsByTagName("supplier")[i].childNodes[0].nodeValue;

        cimoszlop.innerHTML+xmlDoc.getElementsByTagName("to")[i].childNodes[0].nodeValue;
        if
(xmlDoc.getElementsByTagName("delivered")[i].childNodes[0].nodeValue=='true'){
            szallitvaoszlop.innerHTML+'';
        } else {
            szallitvaoszlop.innerHTML+'';
        }
    }
}
}
}
}
}

while(token.toString().length<17)
{
    token+=" ";
}

dateFromString=document.getElementById("fromDate").value;
dateFrom=new Date();
dateFrom.setFullYear(dateFromString.substring(0,4),dateFromString.substring(5,7)-1,dateFromString.substring(8,10));
dateFrom.setHours(0);
dateFrom.setMinutes(0);
dateFrom.setSeconds(0);
dateFrom.setMilliseconds(0);
dateFromWithPadding=dateFromString;

while(dateFromWithPadding.toString().length<17)
{
    dateFromWithPadding+=" ";
}

dateToString=document.getElementById("toDate").value;
dateTo=new Date();
dateTo.setFullYear(dateToString.substring(0,4),dateToString.substring(5,7)-1,dateToString.substring(8,10));
dateTo.setHours(0);

```

```
dateTo.setMinutes(0);
dateTo.setSeconds(0);
dateTo.setMilliseconds(0);
dateToWithPadding=dateToString;

while(dateToWithPadding.toString().length<28)
{
    dateToWithPadding+=" ";
}

req.open("POST",requestAddress,true);
req.setRequestHeader("Content-type", "text/plain");
req.setRequestHeader("x-msg-format", "getOrder");
req.setRequestHeader("x-msg-correlId", username);
req.send(token+"|"+dateFromWithPadding+"|"+dateToWithPadding);
document.getElementById("OrderSearchButton").disabled=true;
document.getElementById("OrderSearchButton").value="Keresés folyamatban";
}
```