IBM Websphere MQ
üzenetsorakoztató middleware

Készítette:

Viczián István

http://jtechlog.blogspot.com

Budapest, 2002 - 2009

Bevezetés

A dolgozat fő témája az IBM Websphere MQ nevű termék, mely egy üzenet központú middleware, azon belül is az üzenet sorakoztató middleware, mely egy kiegészítéssel támogatja a közzétesz és előfizet modellt is. Az IBM Websphere MQ rendkívül elterjedt az iparban, előszeretettel használatos az alkalmazásintegráció terén, mint alapszoftver, amely a pontosan egyszeri biztos üzenetküldést garantálja. Erre épülhetnek különböző más termékek, melyek bróker szerepet tölthetnek be, illetve magasabb szinten munkafolyamat támogatást biztosíthatnak.

Ezzel szemben a dolgozat mégsem termék-specifikus, hiszen az elosztott rendszerek, middleware-ek, tranzakció kezelés és biztonság alapfogalmait is ismerteti.

A dolgozat középpontjában azért áll mégis egy konkrét termék, mert széleskörű alkalmazása, elterjedése és régóta folytatott fejlesztése miatt a legtöbb, gyakorlatban felmerülő problémára megoldást nyújt.

Természetesen alkalmazásintegráció területén más termék is szóba jöhet, ekkor ez a dolgozat alkalmazható összehasonlításra. Nem tartalmaz utalásokat más termékekre, az alapfogalmak megismeréséhez, egy bizonyos követelményszint felállításához azonban kiválóan alkalmas lehet.

A dokumentum megírásának pillanatában (2002. október) az IBM WebSphere MQ 5.3-as verziója a legfrissebb verzió. Azóta kijött a 6-os, majd a 7-es verzió is, melyekben történhettek olyan változások, melyek miatt a dokumentum egy része elavulhatott. Az alapfogalmak, működési mechanizmus azonban megmaradt.

Elosztott rendszerek

Az *elosztott rendszer* fogalma szinte egyidős a számítástechnikával. A hardver, szoftver és a különböző modellek fejlődésével mást és mást értettünk alatta. Sajnos a jelentése még ma sem teljesen kiforrott, viszont annál gyakrabban használt. A fejezet elején megadok egy gyakran használt, eléggé tág definíciót, majd leírok néhány tipikus példát, melyre alkalmazták ezt a fogalmat, így valamiféle áttekintést, összehasonlítási alapot nyújtani. A fejezet végén több osztályozási módot is ismertetek, melyek alapján csoportosíthatjuk az elosztott rendszereket.

Fogalma

Az elosztott rendszer egy több komponensből álló rendszer, amelyek valamilyen kommunikációs mechanizmuson keresztül működnek együtt.
Attól függően, hogy a komponensek milyen jellemzőkkel bírnak, milyen a köztük lévő kommunikációs mechanizmus, osztályozhatjuk az elosztott rendszereket.

Létező rendszerek, fejlődés

Az első igazán elosztott rendszerek csupán távoli hozzáférést tettek lehetővé terminálok számára. Egy szerverhez egyszerre több terminál is csatlakozhatott, olyan problémákat vetve fel, amiket ma is meg kell oldani egy elosztott rendszer tervezésekor és kivitelezésekor. A belső, lokális (LAN), később a nagyvárosi (MAN) hálózatok, majd az egész világot átfogó (WAN) Internet elterjedésével egyre nagyobb igény jelentkezett olyan alkalmazások fejlesztésére, melynek komponensei nem egy szerveren, hanem többön helyezkednek el, megosztva így az erőforrásokat, elosztva a terhelést, biztosítva a kommunikációt, növelve a biztonságot. Mint mindenütt, a szabványoknak itt is nagy szerepe van, különösen a heterogén csomópontok összekapcsolásában, jelentős az International Organization for Standardization nevű szervezet Open System Interconnection hétrétegű modellje nyílt rendszerek összekapcsolásához (OSI ISO modell).
A fentebb említett definíció szerint ide sorolhatnánk a *többprocesszoros* és *virtuális rendszereket* is, melyek problémái és azokra adott megoldásai hasonlóak az elosztott rendszerekéhez, azonban specializáltságuk miatt mégsem itt tárgyalják őket.

Osztályozás

Az elosztott rendszereket több szempont alapján is osztályozhatjuk:

- Kapcsolódás foka szerint. Ezt további két osztályozási szempontra bonthatjuk. Egyrészt a két komponens közötti adatcsere gyorsasága alapján a kapcsolat lehet gyenge kapcsolat (néhány kbit/sec), erős kapcsolat (másodlagos tárolóeszközök átviteli sebességével összehasonlítható sebességűek) és nagyon erős kapcsolat (az átvitel két komponens között majdnem olyan gyors, mint az adattároló és a komponens között). Másrészt a komponensek távolsága alapján lehet nagy távolságú; helyi és többprocesszoros.
- Kapcsolati struktúra szerint. Két komponens között lehet közvetlen kapcsolat. Ekkor vagy minden komponens között különböző kommunikációs szolgáltatás van, vagy egy közös. Két komponens között lehet indirekt kapcsolat is. Ekkor vagy létezik egy centralizált vagy egy elosztott forgalomirányítási mechanizmus. Az utóbbi esetben két komponens között egy vagy több út lehetséges.
- Komponensek függősége alapján. Lehetnek erősen összefüggő komponensek, mikor az egyik komponens műveletének sikeressége a többi komponensen múlik, vagy lehet gyengén összefüggő, mikor egy komponens sérülése nem befolyásolja a többi komponens működését.

- A komponensek közti szinkronizáció alapján. Lehet nem szinkronizált, mikor a komponens a saját sebességén dolgozik, és vár, ha az adat nem elérhető; lehet fix kapcsolat a komponensek feldolgozási sebessége között, és használhatnak közös órát, amit általában a kommunikációs közeg biztosít.

Middleware

A *middleware* a számítástechnikában manapság leggyakrabban használt fogalmak egyike. Ezzel szemben a szó jelentése nem kiforrott, eltérő értelmezésekkel rendelkezik. Összehasonlítva azonban ezeket konkrét igényekkel és megvalósításokkal találkozhatunk, amelyek néhány dologban megegyeznek egymással. Ezeket próbálom kiemelni, mintegy metszetét véve a különböző értelmezéseknek.

Fogalma

A middleware egy olyan elérhető szoftver réteg, mely a heterogén platformok és protokollok hálózati rétege és az üzleti alkalmazás(ok) között helyezkedik el. Leválasztja az üzleti alkalmazásokat bármilyen, a hálózati réteg okozta függőségről, melyet a heterogén operációs rendszerek, hardver platformok és kommunikációs protokollok okoznak.

Szükségessége, kialakulása

A middleware szükségességét jobban megértjük, ha áttekintjük kialakulásának történetét.
Az egyre nagyobb hálózatok kialakulásával egyre gyakrabban merült fel az igény elosztott alkalmazások készítésére, már meglévő alkalmazások integrációjára. Egyik legelterjedtebb és legjobb megoldás a kliens/szerver architektúra, aminek nyilvánvalóvá vált néhány gyengesége az egyre több megvalósítás során. Ezek közül a leglényegesebbek a funkciók szétválaszthatóságának és továbbfejlesztésének képtelensége (ezáltal már meglévő rendszerek összekapcsolása is nehézkes), a skálázhatóság, hordozhatóság, újrafelhasználhatóság hiánya. Ezen problémákra adott megoldások (modellek, kvázi és ipari szabványok, konkrét implementációk) egy háromrétegű architektúra kifejlődéséhez vezettek.
Az első middleware-ek az adatbázis-kezeléshez kapcsolódtak, és egy szabványos nyelvet adtak az adatok elérésére, függetlenítve ezáltal a konkrét adatbázis-kezelőtől az alkalmazást, segítve annak lecserélhetőségét, könnyítve a fejlesztést.
Vállalati alkalmazások fejlesztésekor a legnagyobb probléma a skálázhatóság teljes hiánya volt. Erre adott válasz az alkalmazás komponensekre való bontása, azok elhelyezése az alkalmazásszerveren. Ez egy szoftverréteg a kliens és szerver között, mely gyakran használt funkciókat biztosít, és lehetővé teszi a terheléselosztást. Az üzleti logika a kliens rétegtől átkerül az alkalmazásszerverhez, így alkalmazható a vékony kliens modell, ami azt jelenti, hogy a kliens csakis a megjelenítéssel és a felhasználóval történő kapcsolattartással foglalkozik. Ezzel egy időben az adatbázis-kezelőtől is átkerül az eddig tárolt eljárásokban megvalósított üzleti logika.
Ezen modell elterjedésével, az üzleti logika alkalmazásszerverekbe sűrítésével a fejlesztők mind több és több funkciót, támogatást építettek az alkalmazásszerverekbe, eltakarva a hálózati réteget, helyét magasabb szintű, később szabványosított *alkalmazás programozói interfészek* (API) vették át. Ezek a funkciók az alkalmazások fejlesztését segítették, az API-k pedig a hordozhatóságot.
Tehát a middleware-ekkel kapcsolatosan a következő elvárások fogalmazhatóak meg:

- Szükség van széles körűen elfogadott API-kra, melyek minden funkciót biztosítanak.
- Magasabb absztrakciós szintet nyújtsanak, hogy a fejlesztők az üzleti problémák megoldására koncentráljanak.
- Támogassa a komponens alapú szemléletet.
- Lokális transzparencia, azaz az alkalmazás független attól, hogy egy adott komponens az elosztott rendszeren belül melyik csomóponton helyezkedik el.
- Biztonság, azaz a kommunikáció kódolt legyen, a kliens egyértelműen azonosítható, különböző szerepkörökkel felruházható legyen.

- Az egész rendszer skálázható, azaz a terheléstől függően konfigurálható legyen.
- Néhány esetben elvárható az objektumorientált szemlélet támogatása.
- Elosztott környezetben is működnie kell a hibakezelésnek, hibakeresésnek.

Kategóriái

A middleware-eket öt csoportba oszthatjuk:

- távoli eljáráshíváson (remote procedure call - RPC) alapuló middleware, osztott számítási környezet (Distributed Computing Environment - DCE);
- üzenetközpontú middleware (Message Oriented Middleware - MOM);
- hordozható tranzakció feldolgozó monitor (Transaction Processing Monitors - TPM);
- objektum lekérdező ügynök (Object Request Broker - ORB) beleértve az OLE/COM/DCOM technológiát;
- adatbázis middleware.

Bemutatni részletesen csak az üzenetközpontú middleware-eket fogom, mely kategóriába az IBM Websphere MQ is tartozik.

Távoli eljáráshíváson alapuló middleware, osztott számítási környezet

A *távoli eljáráshívás* egy olyan technológia, mely lehetővé teszi egy alkalmazásnak más gépen lévő alkalmazás eljárásainak meghívását. A távoli eljáráshívás ezáltal egy magasabb absztrakciós szinten helyezkedik el, nincs szükség hálózati protokoll kialakítására, adatkonverzió programozására. A kliens oldali csonk becsomagolja a hívás paramétereit, és eljuttatja a hívással együtt a szerver oldali csonknak. A szerver oldali csonk kicsomagolja a paramétereket, és meghívja az alkalmazás eljárását, mint lokális eljárást. Az visszaadja a visszatérési értéket, melyet a szerver csonk becsomagol, és visszaadja a kliensnek, amit az kibont, és visszaadja a hívó félnek. Innen is látszik, hogy a távoli eljáráshívás szinkron és blokkoló, azaz a kliens (hívó fél) egészen addig blokkolt állapotban marad, míg a szerver (hívott fél) nem válaszol. Azt a folyamatot, mikor a memóriában lévő adatot leképzi a rendszer a hálózaton átvihető üzentté, *leképzésnek* (marshaling) nevezzük. Ennek ellenkezője a *visszaállítás* (unmarshaling), mely az üzenetből újra paramétert készít a lokális eljáráshívás számára.
Ezáltal a távoli eljáráshívás szinte teljesen transzparens a kliens számára, azzal az egy megkötéssel, hogy a kapcsolat kiépítésekor a kliensnek a szervert valahogy meg kell találnia. Ez történhet például címtárszolgáltatáson (registry) keresztül, amiből név alapján konkrét értékeket lehet kinyerni.
A szerver és a kliens közötti kötés lehet statikus, mikor a kapcsolat már fordítási időben eldől, illetve dinamikus, ami független a szerver választásától.
A szerver és kliens közötti interfészt egy *interfész leíró nyelv* (Interface Definition Language – IDL) írja le.
Az Open Group által kifejlesztett *osztott számítási környezet* (Distributed Computing Environment - DCE) egy olyan nyílt szoftver technológia, mely lehetővé teszi különböző platformokon futó elosztott alkalmazások fejlesztését. Célja az alkalmazások hordozhatósága, egy szabvány megteremtése. A DCE szolgáltatások halmazát definiálja, melyeket a gyártók általában az operációs rendszerükbe integrálva biztosítják.

Üzenet központú middleware

Az *üzenet központú middleware*-ek kulcsfogalma az *üzenet* (message), egy bájtsorozat, mely általában két részből áll, a fejlécből, mely az üzenetről tartalmaz információkat, pl. vonatkozhatnak az üzenetek sorrendjére, az üzenet céljára; illetve áll magából az üzenet törzsből, melyet az alkalmazások használnak fel. Az üzenetküldési modellnek fő

tulajdonsága, hogy támogatja mind a szinkron, mind az aszinkron kommunikációt, és inkább az eseményvezérlésre hasonlít, mint a procedurális feldolgozásra.
A jelentős mennyiségű gyártó és termék, a probléma különböző irányokból való megközelítése, szabvány vagy dominens gyártó hiánya miatt jelentős zavar van ezen a területen. Az üzenet központú middleware-eket is több kategóriába sorolhatjuk:

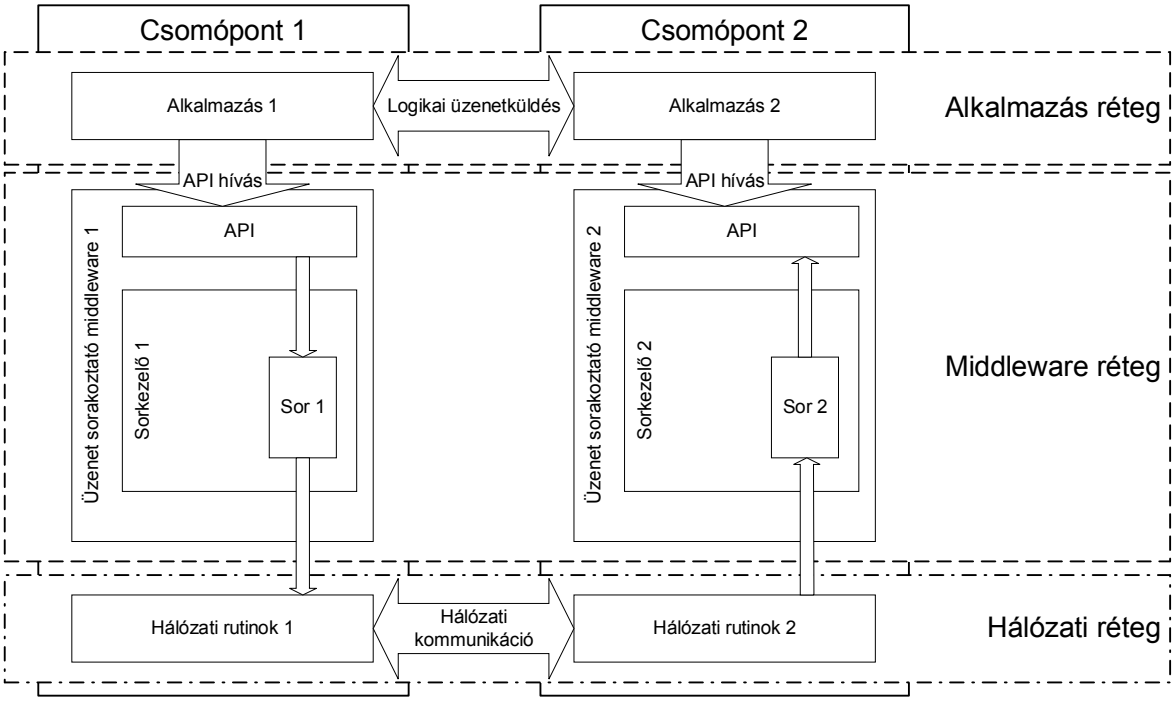
- üzenet továbbadó;
- üzenet sorakoztató;
- „közzétesz és előfizet" megoldások (publish and subscribe).

Üzenet továbbadó middleware

Az üzenet továbbadás egy közvetlen, két program közti kommunikációs modell. Az üzenet továbbadásához egy logikai kapcsolat van a két program között, amit végig fenn kell tartani az üzenetváltás idejére. Ebben a modellben lehetséges a szinkron (az eljáráshíváshoz hasonlatosan a hívó blokkolódik, míg a hívott választ nem ad), illetve az aszinkron (az üzenetet küldő alkalmazás nem várja meg, míg válasz érkezik a címzettől) kommunikációt megvalósítani.

Üzenet sorakoztató middleware

Az *üzenetsorakoztatás* (message queuing) egy indirekt, két program közötti kommunikációs modell. Az üzeneteket a programok *sorba* (queue) helyezik el, és onnan veszik ki. A sorokat sorkezelők (queue manager-ek) kezelik, és nyújthatnak bizonyos többletszolgáltatásokat is, pl. üzenetek prioritásának a kezelése, triggering, terheléselosztás, stb...
Megjegyzem, hogy az itt említett sornál szó sincs arról, hogy csak az először betett üzenetet lehet kivenni a sorból (FIFO). Egyrészt hozzá lehet férni olyan üzenetekhez is, melyek a sor közepén esetleg végén helyezkednek el, és az üzenetek prioritása is egy sorrendet határoz meg, mely nem feltétlenül azonos az időbeli sorrenddel. Ezen kívül elolvasunk egy üzenetet úgy is, hogy az nem feltétlenül kerül ki a sorból. Így teljes szabadságot kapunk az üzenetek feldolgozásának sorrendjében.
Az üzenetsorok és sorkezelők alkalmazásával így nem szükséges az állandó kapcsolat, a kommunikáló programok saját sebességgel futhatnak, tulajdonképpen logikailag függetlenül.
A kommunikáció eme formáját aszinkron kommunikációnak nevezik. Ilyenkor az üzenet küldése, és kézbesítése idő független, nem garantált az adott időn belüli kézbesítés, csak maga a kézbesítés.
A sorkezelők biztosítják az üzenet célba juttatását, és alternatív útvonalakat kereshetnek az elsődleges útvonal kiesése esetén.
Az üzenetsorok és sorkezelők biztosítják a *szolgáltatás minőségét* (quality of service): megbízható üzenettovábbítás (üzenetek nem vesznek el, nem módosulnak), üzenet kézbesítését (hálózat kiesése esetén is megtartja az üzenetet, és a hálózat rendbe jöttekor elküldi) és a csak egyszeri üzenetkézbesítést (a címzett csak egyszer kapja meg, és csakis egyszer).
Az üzenet sorakoztató middleware-ek általában nyújtanak egy sorok és üzenetek kezelésére alkalmas saját API-t, mellyel a middleware sajátságos funkcióit is ki lehet használni, illetve támogathatnak általánosabb, kvázi szabvánnyá vált API-kat, melyeknek használata lehetővé teszi az alkalmazásfejlesztőknek, hogy ne kelljen új API-t megtanulni middleware váltásakor, és ne kelljen annak belső felépítését, működését ismerni, valamint biztosítják a hordozhatóságot.
Az middleware és az API lehetővé teszi, hogy az alkalmazásfejlesztőnek ne kelljen a hálózati protokollal foglalkoznia, nem kell saját protokollt kialakítania a programok közti kommunikációhoz, csupán az API-t kell ismerni, mely nem más, mint eljárások, függvények és konstansok halmaza, ahol az interfész ismert és a megvalósítás rejtve van.



1. ábra Üzenet sorakoztató middleware felépítése.

Más gyártók üzenet központú middleware-ei kapcsolódhatnak egymáshoz, úgynevezett connector-okon keresztül.

Egyes termékek támogatják a *triggereket*, amelyek lehetővé teszik, hogy egy feltétel teljesülése esetén események generálódjanak, miknek hatására bizonyos akciók hajthatóak végre (például egy program futtatása, amennyiben egy sorba öt üzenet érkezik be). Ebben az esetben nem kell a programnak állandóan futnia, jelentős erőforrást takarítva meg ezzel.

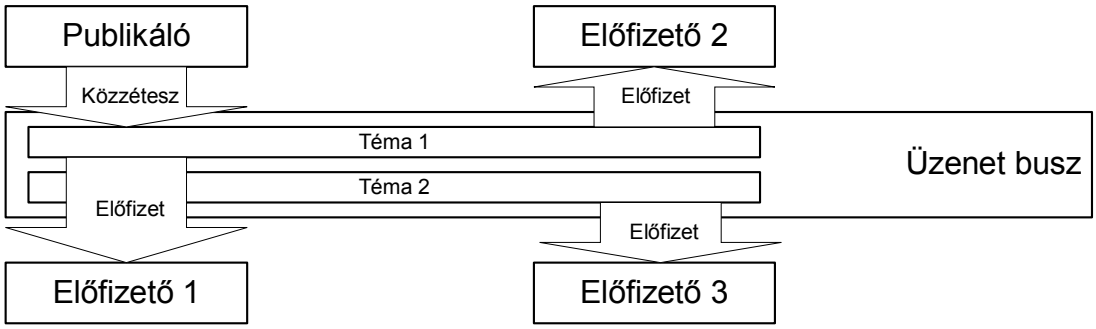
Az üzenetsorok lehetnek *perzisztensek* is, azaz lemezen tároltak (kisebb teljesítmény, de megbízható), illetve csak a memóriában tároltak. Ez utóbbiak nem élik túl az üzenet központú middleware újraindítását.

Az adatok integritása szempontjából fontos, hogy az üzenet sorakoztató middleware-ek ismerjék a tranzakció kezelés fogalmát.

Az üzenet sorakoztató middleware-ek használata ideális elosztott eseményvezérelt alkalmazások fejlesztéséhez, illetve már meglévő alkalmazások integrálásához, és megbízhatósága és tranzakció kezelő volta miatt akár vállalaton belüli, akár vállalatok közötti környezetben is.

„Közzétesz ás előfizet” modell

A „*közzétesz és előfizet*” (publish and subscribe) modell központi eleme egy *üzenetcsatorna*, melyhez mind a közzétevő alkalmazások, mind az előfizetők kapcsolódhatnak. Ez egy alkalmazások közti összeköttetés-mentes modellt biztosít. Az alkalmazások között így nem kell közvetlen kapcsolatnak lennie, így egy új alkalmazás beillesztése illetve leválasztása a rendszerről igen könnyű. Ez megfelelő lazán kapcsolt alkalmazások összekötésére, hiszen az adminisztráció egyszerűbb, az adatforgalom mérsékeltebb és az alkalmazások nem függenek egymástól, tetszőlegesen beilleszthetők és eltávolíthatóak a rendszerből. A közzétevők az információkat kiteszik az üzenetcsatornára egy *témában* (topic), melyek általában hierarchikus szerkezetűek, és az összes előfizető, mely feliratkozott erre a témára, megkapja azt. A „közzétesz és előfizet” modell az interneten elterjedt levelezési listák működéséhez hasonlítható.
A „közzétesz és előfizet” modell ellentéte a pont-pont (point to point) modell, mely esetben egy konkrét alkalmazás a másiknak direkt küld információt.
Meg kell jegyezni, hogy az üzenetközpontú middleware-ek az üzeneteket az adatátviteli protokollok széles skáláján meg tudják valósítani.



2. ábra Közzétesz és előfizet modell.

Elosztott tranzakció feldolgozó monitorok

A jelen fejezet a tranzakció feldolgozó monitorokról (transaction processing monitor – TP monitor) szó, a tranzakció-kezelés alapfogalmaira a későbbiekben térek ki.
Ezen rendszerek egyszerű pooling szolgáltatást nyújtó rendszerekből nőttek ki, melyek az adatbázis kapcsolatok számának csökkentésével növelték a teljesítményt, és csökkentették a terhelést. Ehhez kapcsolódott még rengeteg szolgáltatás, melyekre az üzleti alkalmazások fejlesztésekor szükség van.
Ilyen a tranzakció kezelés, mely kritikus pontja az üzleti alkalmazásoknak. A tranzakció feldolgozó monitor leveszi a terhet az adatbázis-kezelő rendszer válláról, monitorozza, irányítja a tranzakciókat, biztosítva ezzel az adatintegritást. Különböző ipari szabványú interfészek segítségével összekapcsolható más komponensekkel (adatbázis-kezelők, üzenetközpontú middleware-ek, más TP monitorok), biztosítva az elosztott tranzakció kezelés lehetőségét.
Tulajdonságaik közé tartoznak a nagy mennyiségű felhasználó kezelése, processzor terhelésének optimalizálása feladatok priorizálása, ütemezése által, terhelés kiegyenlítés, skálázhatóság, robusztusság.

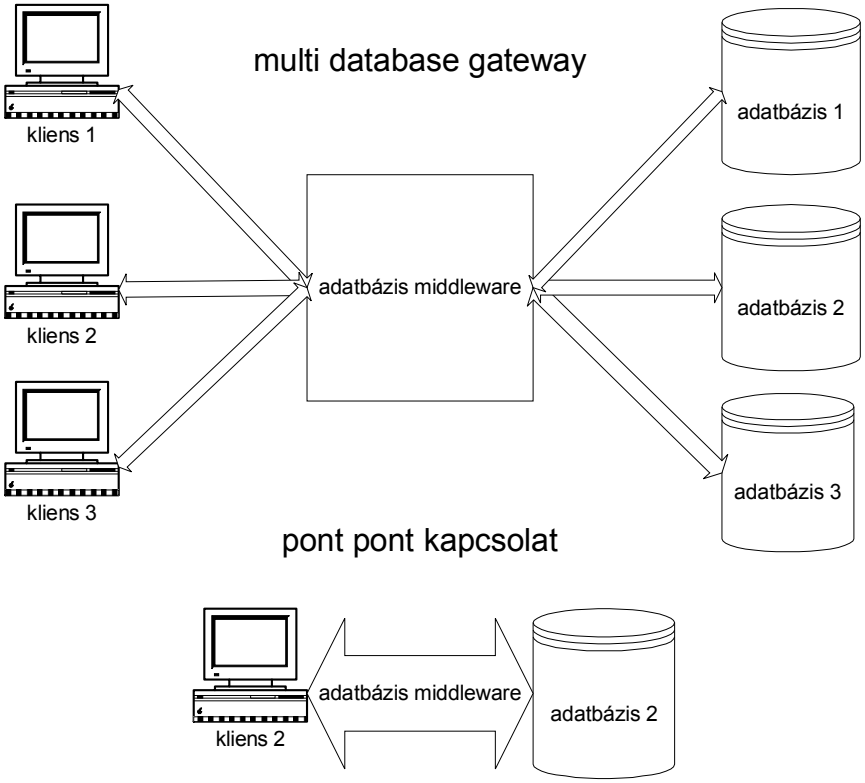
Metódushívás közvetítők

Az objektumorientált middleware-ek, más néven objektumlekérdező ügynökök, vagy metódushívás közvetítők (Object Request Broker – ORB) két csoportba sorolhatóak: az egyik

az OMG CORBA szabványhoz illeszthető, míg a másik a Microsoft OLE/COM technológiája. Mindkettőről temérdek információ áll rendelkezésre a lelkes érdeklődőknek, bemutatásuk túlmutat a dokumentum keretein. Annyit még érdemes megjegyezni, hogy mindkettő az objektumorientált módszertant követi, annak minden előnyével és hátrányával.

Adatbázis middleware

Az adatbázis middleware termékek általános és konzisztens elérést biztosítanak sok adatforráshoz, melyek lehetnek relációs, hierarchikus és objektumorientált adatbázis-kezelők is.
Két architektúra lehetséges. Az egyik a háromrétegű, több adatbázis átjáró (Multi Database Gateway). Ebben az esetben az átjáró a kliens és az adatbázis között helyezkedik el, mindegyike több platformon futhat, és az átjáróhoz több kliens és adatbázis is kapcsolódhat. Az átjáró ekkor analizálja, optimalizálja a kérést, majd lefordítja a cél adatbázis-kezelő nyelvére, növelve ezzel a teljesítményt.
A kevésbé általános architektúra, mikor a kliens közvetlenül kapcsolódik az adatbázishoz az átjárón keresztül, megvalósítva egy direkt pont-pont kapcsolatot.
Az adatbázis middleware alkalmazása ott ajánlott, ahol nem fontos a nagyon gyors válaszidő, és a szinkron, kapcsolat-orientált kommunikáció elegendő.



3. ábra Adatbázis middleware-ek lehetéges architektúrái.

Tranzakció kezelés

A tranzakció kezelés (transaction processing) az üzleti alkalmazások egyik legkritikusabb része. Szerencsére saját rendszer fejlesztésekor nem kell ennek megvalósításával foglalkoznunk, rengeteg megbízható termék van a piacon, szabványosított modellek, interfészek, így elég az alapfogalmakat és a programozói interfészt megismerni ezek használatához. Az előbbihez szeretne hozzájárulni ez a fejezet.

Bevezetés

A *tranzakció kezelés* egy eszköz, mely biztosítja az adatok integritását, mely elvesztésének a veszélye az alábbi esetekben állhat fenn:

- Egyazon alkalmazás hozzáférése ugyanazon erőforráshoz;
- elosztott hozzáférés ugyanazon erőforráshoz;
- egyazon alkalmazás hozzáférése több erőforráshoz;
- több alkalmazás hozzáférése több erőforráshoz.

Megjegyezném, hogy a hozzáférés jelen esetben adatok módosítását jelenti, hiszen egy lekérdezés nem veszélyezteti az adatok integritását. Fontos megjegyezni, hogy a tranzakció kezelés több erőforrást köt le, így kevésbé kritikus helyeken elhagyható annak használata.
A logikailag összekapcsolódó *műveleteket* (operation) *munkaegységekbe* (unit of work) csoportosíthatjuk, meghatározva azt, hogy az egységben lévő műveletek által okozott változások sikeres végrehajtás esetén megmaradjanak, *végrehajtódjanak* (commit), viszont bármely hiba esetén a műveletek előtti állapotok visszaállítódjanak, *visszagörgetésre* kerüljenek (rollback). Ennek biztosítása különösen bonyolult olyan esetekben, ha a műveletek különböző erőforrásokra vonatkoznak, illetve ezek párhuzamosan elérhetők.
A *tranzakció* (más irodalmakban logikai munkaegység – Logical Unit of Work - LUW) egy olyan munkaegység, melyekre igazak az alábbi jellemzők, melyek meglétét az angol kezdőbetűkből képzett ACID tulajdonságnak is nevezik:

- Atomiság (atomicity): a munkaegység műveletei összetartoznak, vagy minden egyes művelet végrehajtódik, vagy hiba esetén a munkaegység futtatása előtti állapotot kell visszaállítani, visszagörgetni. Egyszerűbben megfogalmazva vagy minden műveletet sikerül eredményesen végrehajtani, vagy egyiket sem.
- Konzisztencia (consistency): a tranzakcióban szereplő erőforrásokat a tranzakció konzisztens állapotból konzisztens állapotba viszi tovább. Ez azt jelenti, hogy a párhuzamosan futó tranzakciók eredménye megegyezik ugyanazon tranzakciók egymás után futtatásának eredményével.
- Izoláció (isolation): az eredmény nem függ másik, párhuzamosan futó tranzakció hatásától. Ez két dolgot jelent:
  - Egy tranzakció közbülső (valószínűleg inkonzisztens adatai) nem hozzáférhetőek másik tranzakció számára
  - Két párhuzamos tranzakció nem módosíthatja ugyanazt az adatot
- Tartósság (durability): a tranzakció hatása a végrehajtás után már nem szüntethető meg, a változtatások perzisztensen tárolva lesznek.

Elosztott tranzakció kezelést használó rendszer architektúrája

Abban az esetben, ha egyazon tranzakció műveleteit különböző folyamatok hajtják végre, melyek akár más gépeken is lehetnek, *elosztott tranzakció kezelésről* beszélünk, szemben a lokális tranzakciókkal.
Egy ilyen rendszer három fő komponensből áll:

- Alkalmazás (application): erőforrásokat használ, az üzleti logikát ebben valósítják meg az alkalmazásfejlesztők, a műveleteket tranzakciókba csoportosítja, eldönti, hogy egy tranzakciót végrehajtani vagy visszagörgetni kell-e.
- Erőforrás kezelő (resource manager): kezeli az erőforrást (ahol az adatok perzisztensen tárolva vannak, pl. adatbázis, kommunikációs middleware), részt vesz a kétfázisú végrehajtó folyamat (two-phase commit) megvalósításában. Minden műveletet tranzakcióként kezel, saját hiba esetén automatikus visszagörgeti. A tranzakciók befejezésének eredményét visszaadja a tranzakció koordinátornak.
- Tranzakció koordinátor (transaction manager): lehetővé teszi az alkalmazások számára a tranzakciók használatát, egyedi azonosítókat rendelnek a tranzakciókhoz, figyelik azok állapotát, tartják a kapcsolatot az erőforrás kezelőkkel, levezénylik a kétfázisú végrehajtó folyamatot.

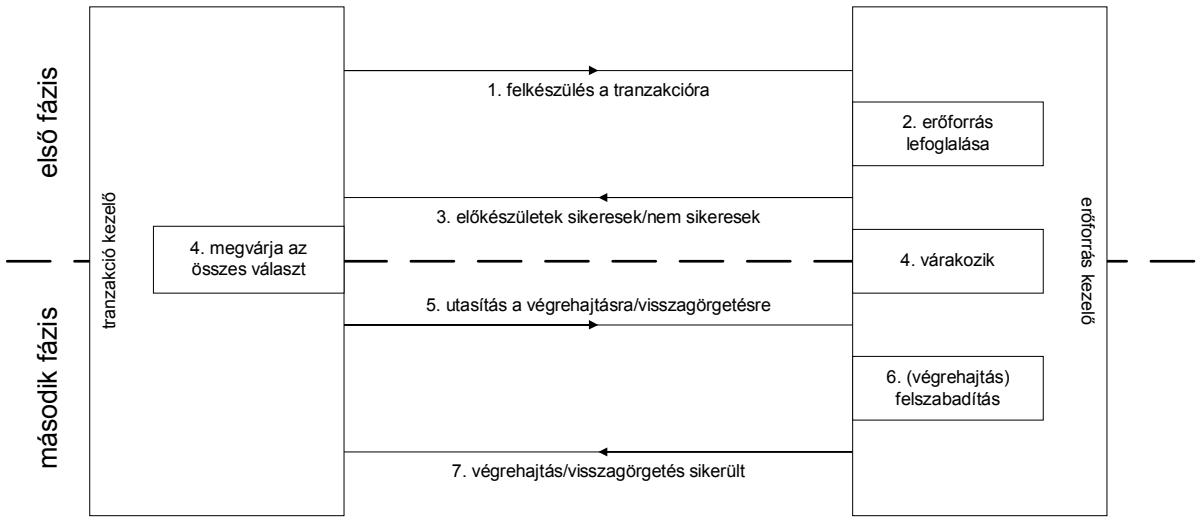
Alapfogalmak

Tranzakció elhatárolás (transaction demarcation) lehetővé teszi elosztott komponensek számára, hogy a műveleteik egy tranzakción belül hajtódjanak végre. Ennek megvalósításának egyik módja a *programbéli elhatárolásnak* (programmatic demarcation). Ekkor a programozó mondja meg, hogy hol kell elkezdeni a tranzakciót, mikor kell commit művelet vagy rollback. A másik megvalósítás komponens alapú tranzakció koordinátoroknál használatos, mikor is a komponensek telepítésekor (deploy) kell megadni, hogy tranzakcióba csoportosuljanak-e a komponens műveletei, vagy sem. Ebben az esetben a tranzakció kezelés az alkalmazásból átkerül a komponens-tárolóra (component container), ezért ezt a technikát *tároló vezérelt elhatárolásnak* (container managed demarcation) is nevezik. Másrészt a tranzakció elhatárolás nem fordítási időben történik (statikus), hanem a telepítéskor (dinamikus).
Amióta a tranzakciókban több komponens és több erőforrás is részt vesz, a tranzakció koordinátornak nyilván kell tartania a tranzakció állapotát. Ez általában *tranzakciós környezet* (transaction context) segítségével történik, ami összekapcsolja az erőforrásokon végzett tranzakciós műveleteket a komponensekkel, melyek meghívják a műveleteket. Ez a környezet általában transzparens az alkalmazások számára, a tranzakció koordinátorok biztosítják.
Az *erőforrás besorolás* (resource enlistment) az a folyamat, mikor az erőforrás kezelők regisztrálják magukat a tranzakció koordinátornál, hogy részt vesznek a tranzakcióban, így a tranzakció koordinátor nyilvántarthassa állapotukat. Szükséges ahhoz, hogy irányítsa az erőforrás kezelők által végzendő műveleteket és a kétfázisú végrehajtó protokoll vezérléséhez.
Attól függően, hogy várunk-e választ a tranzakció eredményéről, megkülönböztetünk szinkron és aszinkron modellt. A *szinkron modellnél* választ várunk a tranzakció kimeneteléről, ennek eredményeképp a rendszereknek azonos rendelkezésre állási jellemzőkkel kell bírniuk, a rendszerek azonos állapotban lesznek. Az *aszinkron modellnél* nem várunk választ, „store and forward” technikát alkalmazunk, azaz egy közbülső tárolón keresztül jutnak el a tranzakcióval kapcsolatos információk a második rendszerhez. Ez azt eredményezi, hogy a rendszerek rendelkezésre állásának képességei különbözhetnek, és a rendszerek állapota átmenetileg különböző lesz. Hiba esetén kompenzációs tranzakciót kell végrehajtani.

A szinkron modellt szorosan csatolt, azonos rendelkezésre-állású elemek között, azonos üzleti rendszeren belül érdemes alkalmazni, míg az aszinkron modellt a lazán csatolt, vagy eltérő rendelkezésre-állású, vagy különböző, integrált rendszerek között.
A szinkron modell esetén alkalmazható az *egyfázisú végrehajtó protokoll* (one phase commit protocol – 1 PC), ami egy szabványos protokoll a tranzakció koordinátor és a besorolt erőforrás kezelők között. A tranzakció végén a tranzakció koordinátor utasítja az erőforrás kezelőket, hogy hajtsák végre a feladataikat. Abban az esetben, ha a kommunikáció megszakad, miután a végrehajtás sikerült, de nem küldte vissza az erőforrás kezelő a választ, az egész rendszer holtpontra jut. Ezt az időintervallumot hívják *bizonytalansági ablaknak*.
Erre a problémára ad részleges megoldást *kétfázisú végrehajtó protokoll* (two phase commit protocol - 2PC), ami biztosítja, hogy egy tranzakcióhoz tartozó összes erőforrás kezelő végrehajtsa, vagy visszagörgesse a kiadott műveleteket. Ez sem szünteti meg a bizonytalansági ablakot, csupán minimalizálja a méretét. Két fázisból áll:

- 1. fázis: a tranzakció végén a tranzakció koordinátor megkérdezi az erőforrás kezelőket, hogy képesek-e a kiadott feladatok (melyeket azok tranzakcióként kezelnek) végrehajtására, vagy nem.
- 2. fázis: ha csak egy negatív válasz is volt, az egész tranzakciót vissza kell görgetni, így minden erőforrás kezelőnek a tranzakció koordinátor kiadja az utasítást, hogy a műveleteket vissza kell görgetni. Ellenkező esetben az erőforrás kezelőket arra utasítja, hogy hajtsák végre a műveleteket.

A döntéshozást a tranzakció koordinátor végzi, de az összes többi résztvevőnek vétójoga van.
A rendszer még biztonságossá tétele érdekében mind a tranzakció koordinátornál, mind az erőforrás kezelőknél naplózás történik, melynek segítségével az egyes komponens teljes összeomlása, majd újraindítása után is biztosított a konzisztencia.
Amint látjuk, a tranzakció koordinátor széthasítja a szinkron tranzakciókat kisebb tranzakciókra, amik aszinkron módon fognak működni aszinkron üzentek által vezérelve.



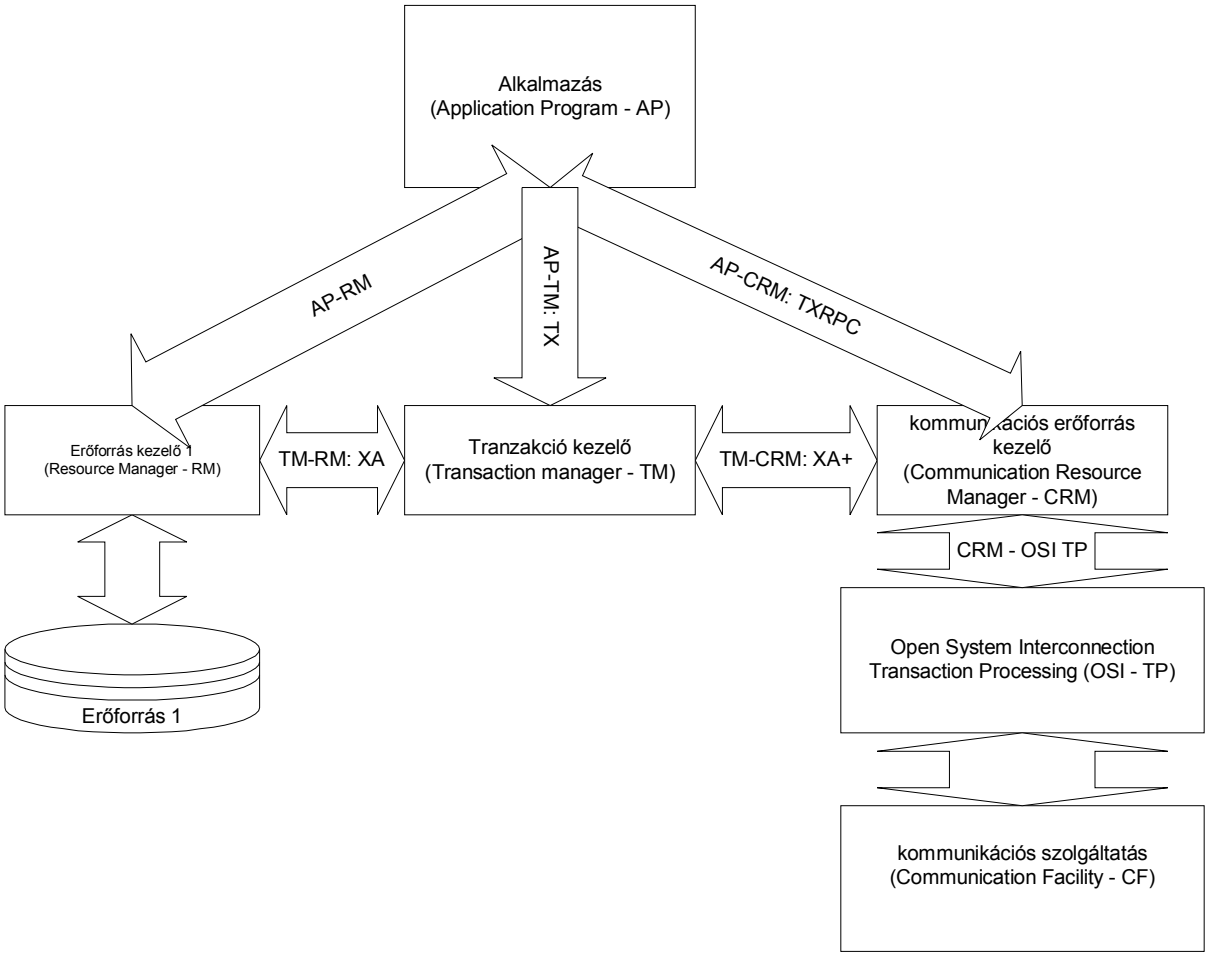
4. ábra Kétfázisú végrehajtó protokoll

Szabványok és technológiák

A szabványoknak ezen a téren is nagyon fontos szerepe van, ugyanis elosztott tranzakció kezelés használatakor mind a tranzakció koordinátor, mind az erőforrás kezelők más gyártók termékei lehetnek, melyeknek kommunikálniuk kell egymással. A következő fejezet az X/Open DTP modelljét mutatja be.
Mivel ez a dokumentum az IBM MQ alapfogalmait, tulajdonságait mutatja be, ezért a tranzakció kezelés terén nem térnék ki egyéb szabványokra vagy technológiákra, ugyanis az elosztott tranzakció kezelés terén az MQ csak az XA interfészt támogatja. Ezért nem részletezném az OMF objektum tranzakció szolgáltatást (Object Transaction Service), amely a CORBA modell kiterjesztése, mely interfészek halmazát definiálja, melyek lehetővé teszik a tranzakció kezelést különböző CORBA objektumokon keresztül. Név szerint megemlíteném még a JTA (Java Transaction API) és JTS (Java Transaction Service) fogalmakat, melyeket a Sun Microsystem Inc. alkotta a Java nyelv tranzakció kezeléssel való kibővítéséhez, a Microsoft Transaction Server-t, mely a COM technológiára épülő komponens-alapú tranzakciós szerver, és az Enterprise Java Beans technológiát, mely specifikál egy keretrendszert vállalati alkalmazások készítéséhez, mely tartalmaz lehetőséget programbéli, mind tároló vezérelt tranzakció kezelésre.

X/Open DTP modell

Az üzleti alkalmazások körében az egyik legelterjedtebb szabvány az elosztott tranzakció feldolgozás (distributed transaction processing – DTP) terén az Open Group által javasolt X/Open DTP modell.
Ez egy olyan szoftver architektúra, mely lehetővé teszi egyszerre több alkalmazás számára a közös erőforrásokhoz való hozzáférést, illetve azt, hogy műveleteiket globális tranzakciókba szervezzék.
Az architektúra a fentebb említett komponenseken kívül még tartalmazza a kommunikációs erőforrás kezelőt (communication resource manager), mely kommunikációs lehetőséget biztosít a különböző tranzakció koordinátorok között.
A modell specifikál hordozható API-kat és rendszerszintű interfészeket, melyek biztosítják az alkalmazás hordozhatóságát különböző X/Open környezetek között, a tranzakció koordinátorok, erőforrás kezelők és kommunikációs erőforrás kezelők lecserélhetőségét és a köztük lévő kommunikációt.



5. ábra X/Open DTP modell.

A következő interfészeket definiálja:

- TX interfész: a tranzakció koordinátor által biztosított interfész az alkalmazás és a tranzakció koordinátor között. Lehetővé tesz tranzakció elhatároló szolgáltatásokat is, mely használatával az alkalmazások tranzakciós műveleteket köthetnek össze egy globális tranzakción belül.

A következő funkciókat biztosítja:

Funkció	**Leírása**
tx_open	Tranzakció koordinátor és a hozzá kapcsolódó erőforrás kezelők megnyitása.
tx_close	Tranzakció koordinátor és a hozzá kapcsolódó erőforrás kezelők lezárása.
tx_begin	Új tranzakció elkezdése.
tx_rollback	Tranzakció visszagörgetése.
tx_commit	Tranzakció végrehajtása.
tx_set_commit_return	Tranzakció végrehajtása.
tx_set_transaction_control	Váltani lehet láncolt és nem láncolt mód között. Láncolt tranzakciók esetén a munka fel lesz osztva darabokra, melyek mindegyike egy külön tranzakció. Egy darab ezután a többi darabtól függetlenül hagyható jóvá, vagy görgethető vissza.
tx_set_transaction_timeout	Tranzakció lejárati idejének beállítása.
tx_info	Információ lekérdezése a tranzakcióról (azonosítója, állapota, stb…).

- XA interfész: a tranzakció koordinátor és erőforrás kezelő közötti kétirányú interfész, melynek funkciói két csoportba sorolhatók:
  Az első csoport funkciói xa_ prefix-szel rendelkeznek, és az erőforrás kezelő biztosítja a tranzakció koordinátornak. Ezen funkciók:

Funkció	**Leírás**
xa_start	Utasítja az erőforrás kezelőt, hogy az alkalmazástól jövő későbbi kérések a paraméterben megadott tranzakcióhoz tartozzanak.
xa_end	Megszünteti a tranzakció és az erőforrás kezelő közti xa_start funkcióval definiált kapcsolatot.
xa_prepare	Előkészíti az erőforrás kezelőt a jóváhagyásra, azaz a kétfázisú végrehajtó protokoll első fázisába lépteti.
xa_commit	Végrehajtja a tranzakciós műveleteket, a kétfázisú végrehajtó protokoll második fázisába lépteti az erőforrás kezelőt.
xa_recover	Visszanyeri az előkészített, és előzőleg végrehajtott vagy visszagörgetett tranzakciókat.
xa_forget	A paraméterként megadott tranzakcióhoz tartozó tranzakciós műveleteket figyelmen kívül hagyja.

A második csoport funkcióit a tranzakció koordinátor biztosítja az erőforrás kezelőnek, és az ax_ prefix-szel kezdődnek. Ezen funkciók:

Funkció	**Leírás**
ax_reg	Az erőforrás kezelő besorozása az elérhető erőforrás-kezelők közé.
ax_unreg	Az erőforrás kezelő eltávolítása az elérhető erőforrás kezelők közül.

- XA+ interfész: az XA interfész kiterjesztése, ami már lehetővé teszi globális tranzakciókról való információk cseréjét azáltal, hogy kapcsolatot biztosít a tranzakció koordinátor és a kommunikációs erőforrás kezelő között.
- TXRPC interfész: az alkalmazás és a kommunikációs erőforrás kezelő közötti interfész, mely tranzakciós távoli eljáráshívást használ.
- CRM-OSI TP interfész: kapcsolatot biztosít a kommunikációs tranzakció koordinátor és az OSI TP (Open Source Interconnection Transaction Processing) szolgáltatás között, mely biztosítja az OSI szolgáltatások implementációjának cseréjét.

Az X/Open DTP modell iparilag nagyon elterjedt. A legtöbb kereskedelmi tranzakció koordinátor termék támogatja a TX interfészt, úgymint a TXSeries CICS/Encina (IBM), Tuxedo, TopEnd (BEA Systems), GIS (AT&T). A Transaction Server (Microsoft) nem támogatja a TX interfészt, de együtt tud működni XA interfészt támogató adatbázis-kezelőkkel. A legtöbb adatbázis-kezelő támogatja az XA interfészt, úgymint az Oracle, Sybase, Informix, Microsoft, IBM (DB2 Connect-en keresztül) adatbázis-kezelője, és az üzenetküldő middleware-ek is, úgymint az IBM MQ és a Microsoft MSMQ Server.

Biztonság

Ebben a fejezetben az *IBM biztonsági architektúráját* (IBM Security Architecture) fogom bemutatni, amit elosztott alkalmazások tervezésekor használhatunk, főleg abban az esetben, ha IBM termékeket is használunk a megoldáskor, jelen esetben az Websphere MQ-t. Ennek bemutatása közben kitérek az alapfogalmakra, melyek nem termék specifikusak, ezzel általános információt szolgáltatva. Az IBM-nél komolyan vették a szabványokat és a nyitottságot más platformok felé, ami különösen fontos elosztott rendszerek fejlesztésekor, alkalmazások integrálásakor, így ez az architektúra is a 7498-2 ISO szabványra épül.

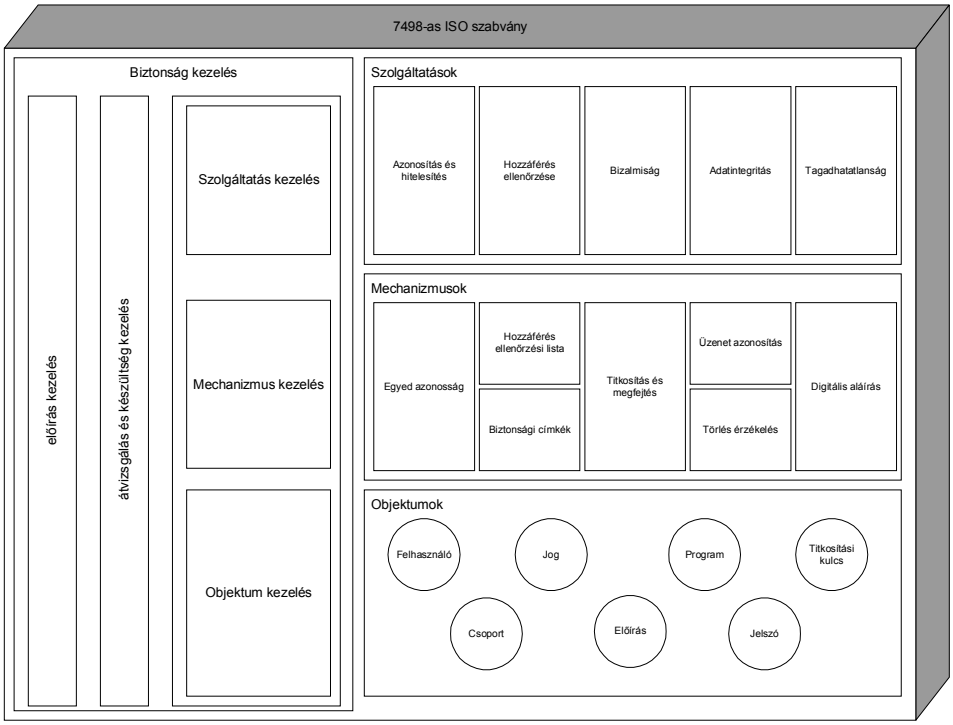
IBM biztonsági architektúra

Az IBM biztonsági architektúra egy modell, mely a biztonsági szolgáltatásokat, mechanizmusokat, objektumokat és felügyeleti funkciókat egyesít eltérő hardver és szoftver platformokon és hálózatokon.
Ebben a fejezetben csak az alapfogalmakra és a modell azon részeire térnék ki, melyek fontosak a Websphere MQ biztonsági fogalmainak megértéséhez.
A biztonsági szolgáltatások (security service) olyan szolgáltatások a rendszeren belül, melyek védik a hozzá tartozó erőforrásokat, érvényesítve a szervezet biztonsági előírásait (security policy). Ezek a következők:

- Azonosítás és hitelesítés (identification, authentication). Az azonosítás egy felhasználó vagy program egyértelmű azonosítása (pl. felhasználónév alapján). A hitelesítés annak ellenőrzése, hogy a felhasználó vagy program tényleg az, aminek állítja magát (pl. jelszó ellenőrzéssel).
- Hozzáférés ellenőrzése (access control). A hozzáférés ellenőrzése biztosítja, hogy egy megadott erőforráshoz melyik program vagy felhasználó hogyan férhet hozzá.
- Bizalmiság (confidentality). A bizalmiság védi az érzékeny információkat attól, hogy illetéktelenek kezébe kerüljenek. Ennek biztosítására lokális rendszeren elég a hozzáférési mechanizmus, de már hálózat esetében ezt magasabb szinten kell kezelni, például titkosítással.
- Adatintegritás (data integrity). Az adatintegritás biztosítja annak észrevételét, ha az adatokat arra illetéktelen módosítja. Ennek forrása lehet hardver vagy adatátviteli hiba, de lehet támadás is. Ez a szolgáltatás nem biztosítja az adatok visszaállítását, csak jelzi az illetéktelen módosítást. Lokális környezetben ez szintén kivédhető egy hozzáférési mechanizmussal, de elosztott környezetben magasabb szintű megoldásra van szükség.
- Tagadhatatlanság (non-repuditation). A tagadhatatlanság biztosítja az adatátvitelkor a bizonyítékát az adat feladójának, kézbesítésének, felterjesztésének és átvitelének. Ennek megvalósítása feltétlenül szükséges az elektronikus kereskedelemnél, és általában digitális aláírással történik. Az elektronikus kereskedelem szabványos üzleti elektronikus adatcserével bonyolódik (Electronic Data Integration – EDI), mely standard üzenetformátumokat és elemeket tartalmaz.

A biztonsági mechanizmusok (security mechanism) olyan technikai eszközök és technológiák, amelyekkel a biztonsági szolgáltatásokat valósítják meg. A biztonsági mechanizmusok például a hozzáférési vezérlési lista (access control list), titkosítás (cryptography), digitális aláírás (digital signature).
A biztonsági objektumok (security objects) tartalmazzák a biztonsággal kapcsolatos információkat a felhasználókról (user), csoportokról (group), jogokról (privilege), előírásokról

(policy), programokról (program), jelszavakról (password), titkosítási kulcsokról (encryption keys), naplókról (audit logs), stb.
A biztonságkezelés (security management) magába foglalja a biztonsági előírások felállítását és érvényesítését, a biztonsági szolgáltatások, mechanizmusok és objektumok kezelését, és a biztonsági környezet átvizsgálását.



6. ábra IBM biztonsági architektúra.

IBM WebSphere MQ (korábban MQSeries) termék

Az üzenetközpontú middleware-ek általános bemutatása helyett egy konkrét kereskedelmi terméket mutatnék be, amely kvázi szabvánnyá vált az iparban, így a felépítése, felvetődött problémák megoldásai jelentős ismeretanyagot hordoznak az üzenetközpontú middleware-ek megismeréséhez.

Bevezetés

Az IBM marketing szakemberei nemrégiben egy teljes arculatváltást bonyolítottak le a szoftverek között, meghatározva négy termékvonalat, melybe az összes IBM szoftvertermék besorolható, megkönnyítve azok megismerését, elhelyezését. Ez a négy irányvonal a WebSphere, mely egy szoftverplatform e-business alkalmazások készítéséhez; a DB2, melynek tagjai az adattárolást valósítják meg; a Lotus, mely csoportmunkát támogató szoftvereket tartalmaz; illetve a Tivoli, amely tagjai rendszerfelügyeletet valósítanak meg. A négy fő trend további szoftvercsaládokat foglal magába fastruktúraszerűen, mely fa levélelemei magukat a termékeket tartalmazzák.

Szoftverkomponensek

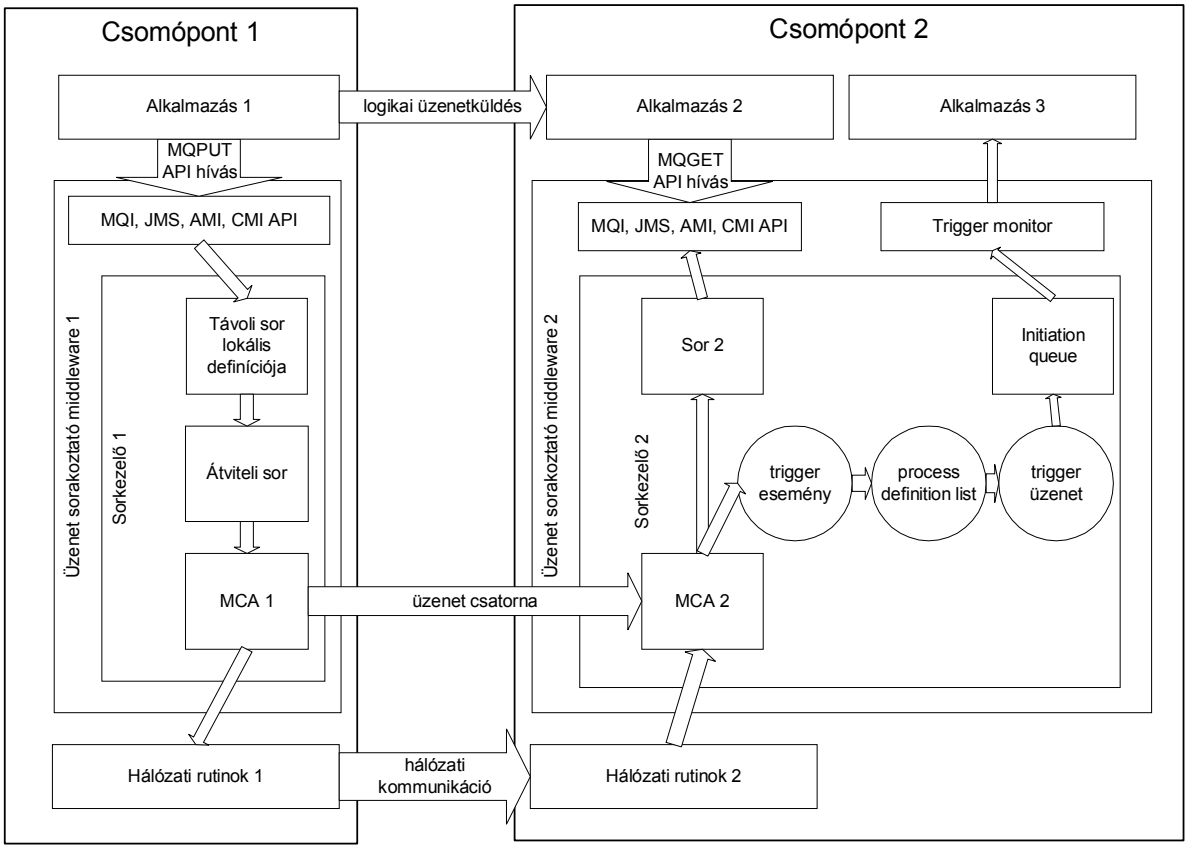
Az IBM MQSeries termékcsalád új neve ebben a besorolásban WebSphere MQ lett, mely jelzi, hogy a WebSphere trendben helyezkedik el, ezen belül is a Business Integration: Application Connectivity termékcsaládban.
Az IBM ebbe a termékcsaládba elhelyezkedő termékeit alkalmazások összekötésére, integrációjára, információcseréjük megvalósítására szánja. A termékcsalád további tagjai a Websphere MQ üzenetküldési szolgáltatásait veszik igénybe, magasabb szintű igényeket elégítenek ki. Ilyen a WebSphere MQ Everyplace (korábban MQSeries Everyplace), mely a mobil eszközökre is kiterjeszti az MQ üzenteküldési képességeit. A Websphere MQ Event Broker (korábban MQSeries Integrator) egy integrációs brókerszoftver, mely az alkalmazások, mint csomópontok közti forgalomirányítást végzi, illetve a csomópontoknál szabályok alapján tartalmi és formai átalakításokat végez. A Websphere Data Interchange egy komplett adatkonverziós és tranzakció kezelő megoldás. A WebSphere MQ Workflow (korábban MQSeries Workflow) egy üzleti munkafolyamat-vezérlő termék. Ide tartoznak még a konnektorok és adapterek, melyek más gyártók szoftvereit kapcsolhatják be az MQ hálózatba.

WebSphere MQ-ról általában

A következő fejezetek magával az MQ termékkel foglalkoznak, mely a fent említett szoftverek alapját is képzi, biztosítva a kommunikációs szolgáltatást. A legtöbb jellemző minden platformon azonos, de egyre specifikusabb részeket kifejtve a platformbeli különbségek nagyok lehetnek. Dolgozatomban a Windows 2000 operációs rendszerre koncentrálok, a platformbeli különbségeket nem említve.
Az MQ üzenet sorakoztató middleware, mely több mint 35 platformon elérhető. Az MQ egy robosztus rendszer, mely biztosítja a biztos, pontosan egyszeri üzenetkézbesítést, amit az alatta lévő hálózati protokoll nem biztosít, még akkor is, ha az általa összekötött programok valamelyike, vagy akár a hálózat összeomlik. A hiba megjavulásakor, az összeomlott komponens újraindításakor az üzenetek kézbesítésre kerülnek. Egyszerű programozási felületet (API-kat) nyújt a fejlesztőknek több nyelven is, így a már meglévő alkalmazások könnyen kapcsolhatók az MQ rendszerbe, anélkül hogy a hálózati protokollra kelljen figyelni, így a fejlesztő összpontosíthat az üzleti logikára. Az általános programozói interfész neve

Message Queue Interface (MQI), mely minden platformon ugyanaz. Az MQ lehetőséget nyújt az üzenet kézbesítésének visszaigazolására is.
Egy MQ rendszer fő komponense a sorkezelő (queue manager), mely a sorokat kezeli. Ez teszi lehetővé, hogy az alkalmazások és folyamatok az MQI segítségével igénybe vegyék az üzenetküldő szolgáltatást. A sorkezelő teszi az üzeneteket a megfelelő sorokba, és biztosítja az üzenetek továbbítását más sorkezelőkhöz.
Az MQ az MQI-n kívül még három API-t támogat, amik nem használják ki annyira az MQ adottságait, viszont ismeretükkel nem kell új API-t megtanulni. Ezek a következők: JMS (Java Message Service), AMI (Application Message Interface) és a CMI (Common Messaging Interface).
Az MQ használatához először definiálni kell sorkezelőt és sort. Ehhez segítséget nyújtanak a megfelelő grafikus adminisztrációs eszközök. De az adminisztrációs tevékenységek egy része programból is elvégezhető.
Az MQI hívások használatához először csatlakozni kell egy sorkezelőhöz, ami visszaad egy kapcsolatkezelőt (connection handle), mely a sorkezelő azonosítója. Minden MQI híváskor át kell adnunk ezt az azonosítót. Egy alkalmazás egyszerre egy sorkezelőhöz kapcsolódhat.
Ahhoz, hogy üzenetet küldjön egy alkalmazás, meg kell nyitni egy sort. A sort meg lehet nyitni írásra, ekkor csak üzeneteket tesz a sorba, olvasásra, ekkor csak üzeneteket vesz ki a sorból, illetve mindkettőre. Ha a megnyitási kérelem sikeres, akkor a sorkezelő egy objektumkezelőt (connection handle) ad vissza, mellyel később hivatkozhatunk a sorra.
Egy üzenet két részből áll, magából az üzenetből, illetve egy üzenetleíróból (message descriptor), melyet a sorkezelő ragaszt hozzá a megfelelő kézbesítés biztosítására.
Persze az alkalmazások és az üzenetsor egyazon gépen is elhelyezkedhetnek, de tipikusabb megvalósítás, mikor az alkalmazások más gépeken futnak. Az első esetben a sorkezelő is azon a gépen fut, ahol az alkalmazások, a második esetben minden MQ rendszerben szereplő gépen futnia kell egy sorkezelőnek, amennyiben a biztos üzenettovábbítás megkövetelt abban az esetben is, ha az alkalmazások közötti csatorna leáll (pl. megszakad a hálózat). Ebben az esetben a sorkezelők között léteznie kell egy üzenet csatornának (message channel), mely a két sorkezelő közötti üzeneteket közvetíti. Legyen az A alkalmazás az üzenetet küldő alkalmazás, a B az üzenetet fogadó alkalmazás. Lokális sor (local queue) az a sor, mely azon sorkezelőben van, amelyhez az alkalmazás kapcsolódik. A távoli sor (remote queue) a többi sorkezelőben lévő sor. Az A alkalmazás nem lokális sorra hivatkozik az üzenet sorba tételekor, hanem egy távoli sor lokális definíciójára (local definition of remote queue). Ilyenkor az a sorkezelő, melyhez az A alkalmazás csatlakozik, egy speciális lokális sorba, átviteli sorba (transmission queue) teszi az üzenetet, amely az üzenet csatornán keresztül átkerül a B alkalmazáshoz tartozó sorkezelőhöz, azon belül is a távoli sorba, ami a B alkalmazás lokális sora, ahonnan kiveheti azt. Az átviteli sor transzparens, a távoli sorra ugyanúgy lehet hivatkozni, mint egy lokális sorra.
Üzenetet kivenni csak lokális sorból lehet, míg betenni lokális és távoli sorba is (az utóbbi esetben transmission queue-n keresztül).
Az üzenet csatorna kezelését az üzenetcsatorna ágens (message channel agent – MCA) végzi, mely az üzenetek átküldéséért, fogadásáért és a konvertálásért felelős.
Az MQ az összes jelentős kommunikációs protokollt támogatja, illetve köztük megfelelő átjárókat biztosít.
Az MQ biztos, aszinkron, pontosan egyszeri üzenetátviteli szolgáltatása még nem elegendő több üzleti alkalmazás összekötéséhez. Ehhez biztosítja a tranzakció kezelést, biztonságos adatátvitelt, triggering mechanizmust, vékony kliens megoldást és terhelés elosztást.
Az MQ képes tranzakció koordinátorként működni, nem csak a sorkezelőket vezérelni, hanem más erőforrás kezelőket is, illetve erőforrás kezelőként más tranzakció koordinátorhoz kapcsolódni, azáltal, hogy az üzeneteket képes munkaegységekbe szervezni.

Elosztott rendszerek esetén a biztonság egy fő szempont. Az MQ esetén biztosítani kell a biztonságos adminisztrációt, a sorkezelők objektumaihoz való hozzáférés ellenőrzést, biztosítani kell a csatornákat. Az MQ minden szint védéséhez lehetőséget biztosít.
A triggering lehetővé teszi, hogy egy alkalmazásnak ne kelljen állandóan futnia, pl. ha nem kap üzenetet, hanem csak akkor induljon el, ha bizonyos feltételek teljesülnek, ezzel is erőforrást felszabadítva. A feltétel teljesülését a sorkezelő figyeli, és ha teljesül, egy *trigger üzenetet* küld egy speciális sornak melynek neve *initiation queue*. Ezt a sort egy speciális alkalmazás figyeli, un. *trigger monitor*, mely az üzenet tartalmától függően elindít egy alkalmazást. A triggerek alkalmazásával így egyetlen folyamat fut több üresjáratú alkalmazás helyett.
A vékony kliens szükséges, ha kis erőforrással rendelkező gépről szeretnénk elérni az üzenetküldő szolgáltatást, és ne kelljen rá sorkezelőt telepíteni. Ilyenkor szükség van egy szerverre, melyen a sorkezelő fut. A kliens ehhez csatlakozik, és egy csatornán keresztül küldi az MQI üzeneteket, illetve kapja vissza a válaszokat. Így a kliens gépen lévő alkalmazások is használhatják az MQ rendszert anélkül, hogy telepítve lenne egy sorkezelő, és az akciók is a szerveren futnak le, jelentős mennyiségű erőforrást takarítva meg. Ennek a megoldásnak a hátránya, hogy nem védett a kliens és szerver közti hálózati kapcsolat megszakításától.
Az adminisztráció könnyítését, és a terheléselosztást fürtökkel (cluster) biztosítja, több sorkezelőt lehet egy fürtbe kapcsolni. Több sorkezelő több sorát is szerepelhet ugyanazon a néven, és akkor a rendszer automatikusan vagy beépített, vagy egyénileg elkészített algoritmus alapján szétosztja az üzeneteket az azonos névvel szereplő sorok között.
Ebben az áttekintő jellegű fejezetben szereplő összes fogalomról, technikáról a későbbi fejezetekben részletesen lesz szó.



7. ábra IBM Websphere MQ felépítése.

Alapfogalmak

Az MQ három alapfogalma az üzenet (message), a sor (queue) és a sorkezelő (queue manager). A következő fejezek ezek tulajdonságait taglalják.

Üzenet

Az üzenet egy bájtfolyam, melynek két része van. Egyrészt az alkalmazás adatai (application data), mely tartalma és felépítése az MQ rendszerbe bekapcsolt programtól függ; illetve az üzenetleíró (message descriptor - MQMD), mely az üzenetről hordoz információt: tartalmazza annak azonosítóját (message id), típusát, az üzenet prioritását, stb... Az üzenet azonosítóját a sorkezelő generálja, és próbálja biztosítani az egyediségét, de akár az alkalmazásban felülbírálható. Az üzenet második része maga az alkalmazásnak szóló adatok, amiket a sorkezelő nem módosít, kivéve, ha konvertálni kell bizonyos adattípusok ábrázolása közti különbség miatt.
Mind a sorkezelő, mind a sor meghatározhat egy maximális üzenet méretet, melynél nagyobb üzenetet sorba tételekor, az alkalmazás hibát kap.
Az MQ négy fajta üzenetet ismer:

- egyszeri (datagram) – egyszerű üzenet, választ nem vár rá a küldő alkalmazás;
- kérés (request) – választ vár a küldő alkalmazás;
- válasz (reply) – maga a válasz üzenet;
- jelentés (report) – eseményt leíró üzenet.

Ezekből az első három alkalmazások közti üzenet, míg a negyediket a sorkezelő küldi az alkalmazásoknak bizonyos eseményekről tájékoztatva azokat.
Abban az esetben, ha kérés és válasz üzenetet használunk, szükség lehet arra, hogy a kettőt összekössük. Ezt javasolt a fejlécben tárolni. Ekkor az eredeti üzenet üzenetleírójának MsgId mezőjét kell átmásolni a válasz üzenet üzenetleírójának CorrelId mezőjébe.
Abban az esetben, ha válasz vagy jelentés üzenetet használunk, meg kell adni az eredeti üzenetben azt a sort és sorkezelőt, melybe a válasz vagy jelentés üzenetet várjuk.
Egy üzenet esetében meg kell adnunk mind az üzenetleíró, mind az adat formátumát, mely fontos az eltérő platformok között utazó üzenetek konvertálásakor. Az üzenetleírónál elég megadni az üzenetleíró karakterkészletét, míg az adatoknál meg kell adni az üzenet formátumát, karakterkészletét és kódolását. Az üzenet formátuma lehet beépített (pl. szöveges), de definiálhatunk sajátot is, ilyenkor nekünk kell megírnunk a konverziós eljárást is. A kódolás a numerikus adatok ábrázolási módját jelenti. Az MQ ezen beállításait az üzenetleíró mezőibe kell beállítani, melyre az MQ nevesített konstansokat biztosít. A konvertálás történhet a küldő és a fogadó sorkezelőnél is, és az MCA végzi.
Az üzenetek prioritást is kaphatnak, amely egy pozitív egész szám lehet. Egy sornál meg lehet adni, hogy az üzeneteket idősorrendben vesszük ki, vagy előbb a prioritást vizsgáljuk. Egy sornál beállíthatjuk az alapértelmezett prioritást, azaz ha egy üzenetnek nem állítjuk be, akkor milyen prioritással rendelkezzen, illetve egy maximum prioritást. Egy alias vagy távoli sornak más értéket is megadhatunk, és annak értékei fognak érvényesülni, amelyiket megnyitottuk.
Az üzenetek csoportokba (group) rendezhetőek, amik az üzeneteket logikailag összekapcsolja. Van egy egyedi azonosítója (group id), melyet a sorkezelő generál, így egy üzenetet a csoportja azonosítójával, és a csoporton belüli pozíciójával lehet azonosítani.
Nagy üzeneteket darabolhatunk (segmentation) is. Az üzenet darabot egy csoportazonosító, egy csoporton belüli pozíció és a szegmensen belüli pozíció azonosítja. Ebből látjuk, hogy mind a csoporton, mint egy feldarabolt üzeneten belül számít a sorrend.
Az üzenetek lehetnek perzisztensek, ami azt jelenti, hogy mind a napló fájlokban, mind adat fájlokban megjelennek. A sornál meg lehet adni egy alapértelmezést, és a külön nem beállított üzeneteknél ezt veszi.

Üzenetet ki lehet venni a sorból nem csak a prioritás és/vagy sorrend alapján, de akár az azonosítója vagy CorrelId mezője alapján. Illetve felhasználható a csoportazonosító is. Ha több üzenetet is visszaad, akkor eldönthető, hogy idő vagy prioritás szerint legyenek-e sorrendbe állítva.
Abban az esetben, ha a sorkezelő nem tud egy üzenetet betenni az adott sorba, több választási lehetőség is van. Megpróbálhatja később betenni, visszaadhatja az üzenetet a küldőnek és beteheti a dead-letter queue-ba.
Abban az esetben, ha az üzenetek (egy vagy több) egy munkaegységbe tartoznak, és egy üzenet feldolgozásakor hiba keletkezik, és visszagörgetésre van szükség, akkor az ehhez a munkaegységhez tartozó összes kivett üzenet visszakerül a sorba. Igazából kivételkor csak megjelöli a sorkezelő kivettnek, és commit hatására kerül ki fizikailag, rollback hatására a jelölés megszűnik. Ha az üzenet jelölt, más alkalmazás nem veheti ki a sorból. Ha egy üzenet olyan hibát tartalmaz, melyet az alkalmazás nem korrigál, végtelenciklus alakulhat ki. Erre megoldás az üzenetek BackoutCount mezője, mely tartalmazza, hogy hányszor lett visszagörgetve az adott munkaegység, melybe az üzenet tartozik.

MQ objektumok

Az MQ a következő objektumokat ismeri:

- sorkezelők;
- sorok;
- névlisták (name list), az 5-ös verziótól;
- Folyamat definíciók (process definition)
- Csatornák

Minden objektumnak van egy objektum leírója (MQOD – object descriptor). A dinamikus sorok (alkalmazásból létrehozott sorok) kivételével ezen objektumokat definiálni kell a következő módok egyikén:

- PCF parancsok;
- MQSC parancsok;
- MQSeries Explorer vagy MQSeries Web Administration felületeken keresztül.

Sor

A sorok tárolják az üzeneteket. Ezek alkalmazástól függetlenül léteznek és névvel rendelkeznek. Egy sorkezelőn belül a nevük egyedi. Lehetnek a memóriában (ha csak ideiglenes), lehetnek merevlemezen (perzisztens sorok) illetve kisegítő tárhelyen (mentés és visszaállítás esetében). Minden sornak csatlakoznia kell egy sorkezelőhöz, mely vezérli azt.
A soroknak az alábbi tulajdonságaik vannak:

- alkalmazás vehet-e ki üzenetet a sorból (get enabled);
- alkalmazás tehet-e üzenetet a sorba (put enabled);
- a sorhoz egy vagy több alkalmazás tartozik;
- az üzenetek maximális száma, mely tárolható a sorban (max. queue depth);
- az üzenetek maximális mérete (max. message size).

Speciális sorok a következőek lehetnek:

- Elosztott sort csak OS/390 rendszeren definiálhatunk, és ez egy olyan sor, mely több sorkezelőhöz is tartozik
- Alias sor (alias queue) olyan sor, mely nem létezik külön fizikailag, csupán egy referencia egy másik sorra, mely lehet lokális vagy távoli is. Ugyanazon sorra így más névvel lehet hivatkozni. Az alias sor más tulajdonságokat is felvehet, mint az eredeti sor.

- A modell sor (model queue) egy olyan sor, mely sablonként (template) szolgál új sorok létrehozásához.
- A dinamikus sor olyan sor, melyet egy alkalmazásból, ideiglenesen hozunk létre egy modell sorból, melynek átveszi a tulajdonságait. Akkor keletkezik, ha az MQOPEN-t modell sorra hívunk, és ekkor létrejön egy új példány.
  A dinamikus sornak két típusa van:
  - Ideiglenes: nem lehet megosztott sor, csak nem perzisztens üzeneteket tartalmazhat, nem állítható vissza sorkezelő újraindítása után, a létrehozó alkalmazás terminálása vagy lezárás után megszűnik.
  - Állandó: tartalmazhat perzisztens üzeneteket is, hiba esetén visszaállítható, alkalmazás által törölhető.

  A dinamikus sorokat pl. akkor érdemes használni, ha nem szükséges, hogy az alkalmazás terminálása után megmaradjon, pl. ide várhatnak választ request üzenetekre.
  Ha több alkalmazás is használja a dinamikus sort, és az egyik törli azt, akkor a valós törlés eltolódhat egész addig, míg az utolsó üzenet nem hagyódik jóvá.
- Fürt sor olyan sor, mely speciális szereppel bír egy fürtön belül, minden sorkezelő elérheti azt külön üzenetcsatorna definiálása nélkül (automatikusan létrehozódik).

Lokális sorok típusai:

- Átviteli sor (transmission queue): az üzeneteket tartalmazza a lokális sorkezelőben, melyeket továbbküld a távoli sornak. Egy üzenet több sorkezelőn, így több átviteli soron is keresztülmehet (multi hopp). Egy sorkezelőben több átviteli sor is lehet, azonos távoli sorkezelőhöz akár több sor is tartozhat.
- Initiation queue: alkalmazás automatikus indításához szükséges egy bizonyos feltétel teljesülése esetén. Trigger üzenetek kerülnek bele.
- Dead-letter queue: nem kézbesíthető üzenetek sora. Egy fejléc is kapcsolódik az ide került üzenetekhez (dead-letter header - MQDLH), ami tartalmazza az eredeti célt, az üzenet idekerülésének okát és idejét. Alkalmazások közvetlenül is használhatják.
- System command queue: megfelelően hitelesített alkalmazások MQ parancsokat (programmable command format - PCF, MQ parancsok - MQSC, vezérlő parancsok - CL) küldhetnek.
- System default queue: nem dinamikus sor létrehozása esetén az MQ ebben a sorban tárolja a sor definíciókat (queue definition)
- Channel queue: elosztott sor kezeléséhez való sor.
- Event queue: sorkezelők vagy csatornák által kiváltott események üzeneteinek tárolására szolgáló sor.

Sorok tulajdonságai:

- név;
- típus;
- leírás;
- vehetnek-e ki a programok üzeneteket;
- tehetnek-e be a programok üzeneteket;
- üzenetek alapértelmezett prioritása;
- üzenetek alapértelmezett perzisztenciája;
- létezik-e bejegyzés ehhez a sorhoz a névszolgáltatóban.

Sorkezelő

A sorkezelő egy program (Windows 2000 rendszeren szolgáltatás), mely a sorokat kezeli, és lehetővé teszi az alkalmazásokat, hogy MQI interfészen keresztül sorkezelő műveleteket

hajtsanak végre. A sorkezelő transzparens az alkalmazások számára. A sorkezelő lehetőséget biztosít

- a sorkezelő objektumainak tulajdonságának változtatására;
- bizonyos eseményeket generál, ha megfelelő feltételek teljesülnek (triggering);
- az üzeneteket az alkalmazás által meghatározott sorba teszi, ami ha sikertelen, megfelelő visszatérési értéket ad vissza.

Minden sor egy sorkezelőhöz tartozik, és ennek a sorkezelőnek a lokális sora (local queue). Az a sorkezelő, melyhez az alkalmazás kapcsolódik, az alkalmazás lokális sorkezelője (local queue manager). Egy távoli sor (remote queue) olyan sor, mely nem a lokális sorkezelőn van. Minden sorkezelő, mely nem azonos a lokális sorkezelővel, távoli sorkezelő (remote queue manager).
Bizonyos operációs rendszereken egy gépen egyszerre több sorkezelő is futhat.
A sorkezelők lekérdezhető, de nem módosítható tulajdonságai:

- név;
- platform;
- rendszervezérlő parancsokból mennyit támogat (command level);
- maximális prioritás, amit rendelhet az üzenetekhez (pl. 9 esetén 0-9-ig oszthatja ki a prioritásokat);
- karakterkészlet;
- üzenetek maximális hossza;
- támogatja-e a szinkronizációs pontot.

A sorkezelők változtatható tulajdonságai:

- szöveges leírása;
- trigger üzenetek időintervallumának korlátozása;
- dead-letter queue neve;
- alapértelmezett transmission queue neve;
- maximális nyitott kapcsolatok száma;
- különböző jelentés üzenetek típusainak engedélyezése vagy tiltása;
- egy munkaegységen belül nem végrehajtott üzenetek maximális száma.

Névlista

Csak az 5-ös verziók támogatják a névlistákat, mely egy olyan objektum, mely fürt és sor neveket tartalmazhat.

Process definition

Automatikusan indítható alkalmazások tulajdonságait tartalmazza.

Csatorna

A csatorna kommunikációs útvonal. Két fajtája van: üzenet csatorna és MQI csatorna. Az üzenetcsatorna üzenetek továbbítására szolgál két sorkezelő között, ami egyirányú, így az oda-vissza kommunikációhoz kettő kell belőle. Az MQI csatorna a kliens-szerver konfigurációnál fordul elő, és kétirányú. Alkalmazások számára a csatornák láthatatlanok.

Fürt (cluster)

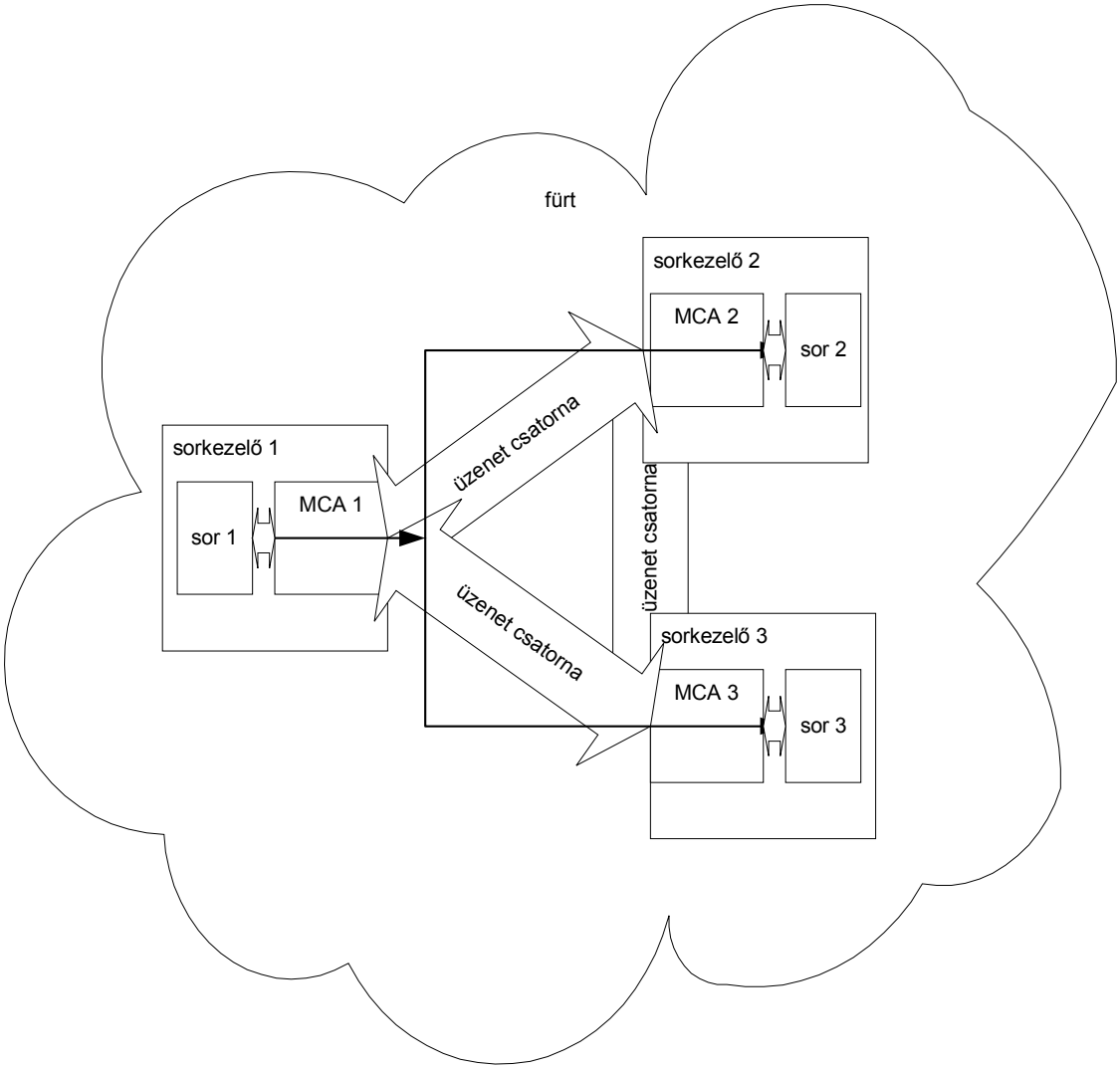
A fürt sorkezelők összessége, mely megkönnyíti azok adminisztrációját. Egy sorkezelő eshet fürtön kívül is, illetve tartozhat egy vagy több fürthöz is. A fürtnek két fő előnye van:

- Adminisztrálás szempontjából a fürtön belüli sorkezelők között automatikusan kialakulnak az üzenetet-csatornák, nem kell őket manuálisan létrehozni. Illetve

bármely sort minősíthetünk fürt sornak (cluster queue), így elérhetővé válik a fürt összes sorkezelőjének, anélkül, hogy lokális definíciót kellene rá konfigurálnunk. A megosztott sort publikus sornak is nevezhetünk (public queue), míg a többit privát sornak (private queue);

- A fürt segítségével megvalósítható a terheléselosztás.

Egy fürt legalább egy sorkezelőjét tárnak (repository) kell nyilvánítani, amely tartalmazza az információkat a fürtről. Biztonság érdekében ajánlott több sorkezelőt is kijelölni erre a feladatra, a több tár közti szinkronizációt az MQ automatikusan elvégzi. Erre két speciális csatornát használ, melynek nevei fürt küldő és fogadó csatorna (cluser sender, cluster receiver). Ha több sort is ugyanazon a néven fürt sornak definiálunk ugyanazon a fürtön belül, akkor az MQ ekvivalensnek tekinti őket. Ekkor az ilyen nevű sorba szánt üzeneteket egy terheléselosztó algoritmust (load-balancing algorithm) válogatja szét a sorok között, tehát mindig csak egy példányba kerül be. A terheléselosztó algoritmus lecserélhető saját implementációra, mely egy cluster workload exit nevű program. Ha tudjuk, hogy melyik sorba akarunk tenni egy üzenetet, akkor a sorkezelő nevét és a sor eredeti nevét használjuk címzéskor.



8. ábra Fürt működése terheléselosztáskor.

Tranzakciós támogatás

A munkaegységnek két fajtája lehet: *lokális munkaegység* (local unit of work), illetve *globális munkaegység* (global unit of work).

A lokális munkaegység esetén csak az MQ sorkezelőjéhez tartozó erőforrásokat kell módosítani. Ebben az esetben a szinkronizációs pont koordinációt a sorkezelő végzi, ami egy egyfázisú végrehajtó folyamat.
Globális munkaegység esetén az erőforrások más erőforrás kezelőkhöz is tartozhatnak, mint pl. XA-kompatibilis adatbázis-kezelő. Ebben az esetben kétfázisú jóváhagyó folyamatot kell alkalmazni, amelyet vezérelhet a sorkezelő maga, ekkor tranzakció koordinátorként működik, vagy akár egy külső, XA-kompatíbilis tranzakció koordinátor, ekkor az MQ egyszerű erőforrás kezelőként működik.

Vezénylő események (instrumentation events)

A vezérlő események segítségével lehet monitorozni a sorkezelők műveleteit. A vezérlő események speciális üzeneteket, úgynevezett esemény üzeneteket (event message) generálnak, ha a sorkezelő bizonyos feltételek teljesülését érzékeli. Az esemény üzenetek egy eseménysorba (event queue) kerülnek.
Abban az esetben, ha az eseménysor egy távoli csomóponton lévő sorkezelőben van definiálva, az MQ hálózat összes sorkezelőjét egy független gépen monitorozhatjuk.
Az eseményeket a következő kategóriákba sorolhatjuk:

- Sorkezelőhöz tartozó esemény (queue manager event): sorkezelőn belüli erőforrásokhoz tartozó események. Pl. egy alkalmazás meg akar nyitni egy sort, de nincs hozzá jogosultsága.
- Teljesítménytől függő esemény (performance event): ha az adott erőforrás elér egy bizonyos határértéket. Pl. a sor megtelik, így nem szolgálja ki az alkalmazást, mely egy újabb üzenetet rakna bele.
- Csatornához tartozó esemény (channel event): csatornához tartozó feltétel teljesülésekor váltódik ki. Pl. esemény generálódik, ha a csatorna elindul vagy leáll.

Triggering

Egy alkalmazás elindulhat, ha üzenet érkezik egy sorba, illetve bizonyos számú esemény összegyűlt. Ekkor az üzenet(ek) feldolgozása után le is állhat. Ekkor az alkalmazásnak nem kell tétlenül futnia, jelentős erőforrást szabadítva fel ezzel.

Parancs halmazok

A parancs halmazok (command set) objektumok (sorkezelők, sorok, processzek és csatornák) létrehozására, törlésére, elindítására, leállítására, tulajdonságainak lekérdezésére, módosítására használhatók.
Típusai:

- Vezérlő parancsok (CL – command language): Windows esetén parancssorba begépelhető parancsokat jelentenek. Ennek is három típusa van:
  - Sorkezelő parancsok: létrehozására, törlésére, elindítására, leállítására.
  - Csatorna parancsok: elindítására, leállítására, inicializálásra.
  - Segédprogramok.
- MQSC (MQSeries commands): Windowsban a runmqsc segédprogramok kell indítani, és ide lehet begépelni őket.
- PCF (programmable command format): alkalmazásokban használatos adminisztrációs teendők végrehajtásához, annak automatizálásához és centralizálásához.
- MQAI (MQSeries Administration Interface): egy egyszerűbb felület adminisztrációs programok írásához.

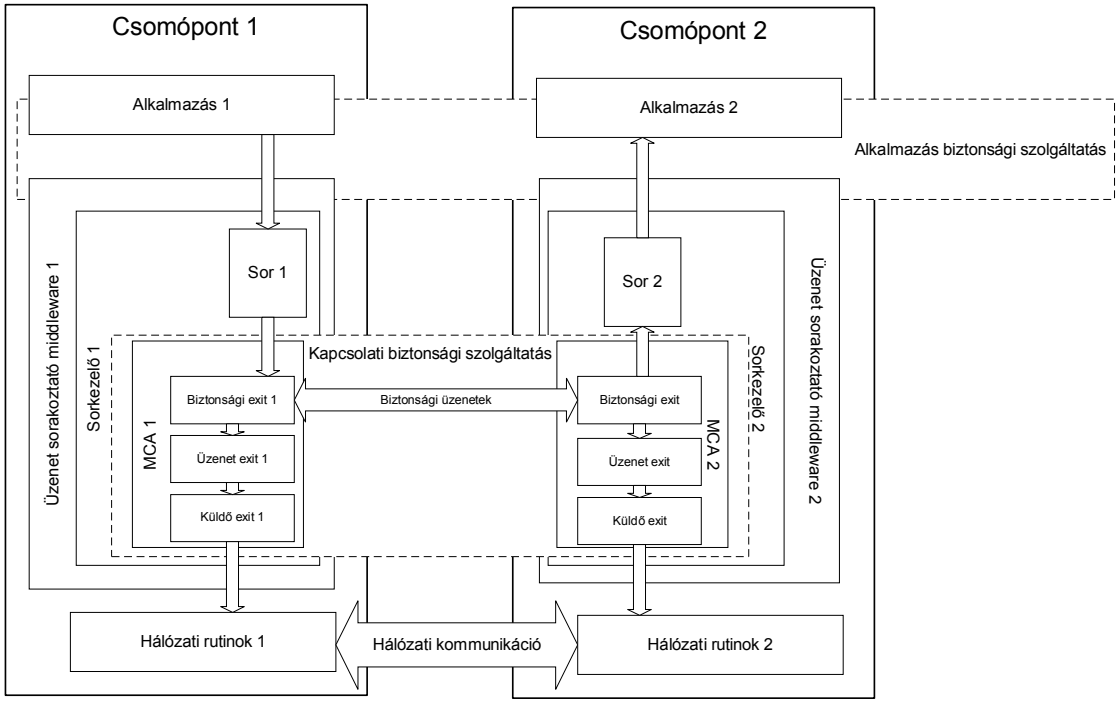
Közzétesz és előfizet modell

A közzétesz és előfizet modellt az MQ 5.3 alapesetben nem támogatja, használatához le kell tölteni hozzá egy kiegészítő csomagot, melynek neve MQSeries Publish/Subscribe, melyet a SupportPac MA0C tartalmaz.

Biztonság

A következő fejezet leírja, hogy milyen biztonsági szempontokat kell figyelembe venni egy MQ-ra épülő rendszernél.
Alapvető szempontok közé tartozik, hogy az MQ adminisztrálásakor, objektumokhoz való hozzáféréskor megfelelő jogokkal kell rendelkezni, így van ez a csatorna biztonságánál is. Windows NT rendszeren az adminisztráláshoz megfelelő jogosultsággal kell rendelkezni adminisztrációs parancsok végrehajtásához és az MQ Explorer és Webshere MQ Services Management Consol-ba beépülő modulok használatához. A sorkezelő, sor, folyamat és névlista objektumok MQI hívásokkal és PCF parancsokkal történő hozzáférésekor is az ezeket használó alkalmazáshoz tartozó felhasználói azonosító (userid) szerint történik az azonosítás, és ezeket az objektumokat az MQ védi. A csatornánál a MCA-hoz tartozó felhasználói azonosító szerint történik az azonosítás, és ennek a felhasználónak jogosultságokkal kell rendelkeznie bizonyos erőforrások elérésére. Mind az adminisztátori, objektumhozzáférési és MCA-hoz kapcsolatos jogok ellenőrzését az MQ biztosítja a hozzáférés vezérlési (access control) mechanizmussal.
További szempontokat kell figyelembe venni fürtök, közzétesz és előfizet modell és az Websphere MQ Internet pass-thru használatakor. A fürtök használatakor különösen figyelni kell arra, hogy egy sorkezelőnek csak kiválasztott sorkezelők küldhessenek üzenetet, a távoli sorkezelők csak kiválasztott felhasználói küldhessenek üzenetet, és az alkalmazások csak a kiválasztott távoli sorkezelőknek küldjenek üzenetet. Csak a fürtök használatakor kell figyelni arra, hogy csak kiválasztott sorkezelők csatlakozhassanak a fürthöz, és a továbbiakban nem szükséges sorkezelőket el kell távolítani a fürtből. A közzétesz és előfizet modell használatakor a közzétevő és előfizető alkalmazásoknak hozzá kell férnie azon sorokhoz, amelyek a brókerrel való kommunikációt biztosítják. Az Internet pass-thru szintén egy MQ alapú alkalmazás, melyet a SupportPac MS81 tartalmaz, és lehetővé teszi, hogy két sorkezelő között kapcsolat létesüljön az Interneten keresztül, közvetlen TCP/IP kapcsolat nélkül. Adattovábbításra a HTTP protokollt használja, így nem akadhat fenn a tűzfalon, és SSL használatával a kommunikáció titkosítható.
A biztonságnál kell még beszélnünk a kapcsolati szintű (link level security) és alkalmazás szintű (application level security) biztonságról. Az előbbihez olyan biztonsági szolgáltatások tartoznak, melyeket direkt vagy indirekt az MCA vagy kommunikációs alrendszer hívhat a kapcsolódási pontjuknál. Például az MCA a másik MCA-hoz való kapcsolódáskor hitelesítheti azt, az átvitt üzenet titkosítható vagy a célállomás megvizsgálhatja, hogy az üzenetet nem módosították-e az adatátvitel közben. Az utóbbihoz olyan szolgáltatások tartoznak, melyeket direkt vagy indirekt az alkalmazás vagy a sorkezelő hívhat a kapcsolódási pontjuknál. Például az üzenetbe betehető, hogy mely felhasználó veheti ki azt a sorból, illetve ezen a szinten is megoldható az üzenet kódolása és dekódolása, valamint az üzenet módosításának ellenőrzése.
Több jelentős eltérés is van a kapcsolat szintű és alkalmazás szintű biztonsági szolgáltatások között, melyeket érdemes átgondolni, és ezek alapján választani a két technika között. A kapcsolat szintű biztonság nem védi az üzeneteket a sorban, csakis az átvitel közben védi azokat. Erre nincs is szükség akkor, ha az alkalmazások üzeneteket csak egy sorkezelőn keresztül cserélnek. Az alkalmazás szintű biztonság ezzel szemben több erőforrást használ, több adminisztrációs tevékenységgel jár, és nem biztos, hogy minden platformon elérhető, vagy ugyanolyan könnyen megvalósítható. Ezek mellett alkalmazásszintű biztonságnál nem

lehet megvalósítani az MCA-k kölcsönös hitelesítését, nem lehet levédeni az átviteli csatorna fejlécét és az MQI csatornán közlekedő paramétereket sem lehet titkosítani. Figyelni kell arra is, hogy ne csak a célállomás tudjon egy kódolt üzenetet megfejteni, mert akkor a dead-letter sorba került titkosított üzeneteket nem lehet feldolgozni.
A kapcsolat szintű biztonságot channel exit programokkal (channel exit program) lehet megoldani, melyeket az MCA hív meg jól definiált pontokon. Az MQ lehetőséget biztosít saját programok megírására is. A channel exit programokat 4 csoportba lehet osztani: biztonsági, üzenet illetve küldő és fogadó. A biztonsági channel exit programot az MCA-k kölcsönös hitelesítésére lehet felhasználni, és a kapcsolat inicializálásakor hívódik meg. A két sorkezelőben lévő biztonsági channel exit programok között a kommunikáció biztonsági üzenetekkel (security message) valósul meg. Az üzenet channel exit programok az üzenetek átviteli sor fejlécéhez és az alkalmazás adataihoz férnek hozzá. A küldő és fogadó channel exit programok az adatfolyamhoz férnek hozzá, az egyiknek a küldő, a másiknak a fogadó oldalon kell elhelyezkedniük.
A Tivoli Policy Director egy önálló termék, mely alkalmazás szintű biztonságot biztosít egy MQ alapú rendszer számára, mely nyilvános kulcsú titkosítást alkalmaz. A Tivoli Policy Director az IBM biztonsági architektúra szolgáltatásainak minden szintjét támogatja.
Az MQ lehetőséget biztosít saját alkalmazás szintű biztonság megvalósítására is, melyhez két exit-et, API és API-crossing exit-et nyújt. Mindkettő egy program modul, mely monitorozza és megváltoztatja az MQI hívások működését.
Az MQ az 5.3-as verziótól kezdve támogatja az Secure Socket Layer-t (SSL), mely egy ipari szabványú protokoll nem biztonságos hálózaton történő adatcserére.



9. ábra Alkalmazás és kapcsolat biztonsági rétegek.

MQ konfigurációk

A legegyszerűbb konfiguráció, mikor egy sorkezelő van egy gépen, és az alkalmazások ehhez kapcsolódnak, és így hozzáférhetnek az összes sorhoz. Bonyolultabb, mikor több sorkezelő is van, és az üzenetek továbbítását a két sorkezelő között kell megoldani. A sorkezelők lehetnek ugyanazon gépen vagy akár más gépeken is. Ekkor egy üzenetcsatornát (message channel) kell létesíteni a sorkezelők között. Az üzenet csatornákat úgynevezett ágensek (message channel agent) kezelik. Abban az esetben, ha a rendszerben több sorkezelő is van, akkor nem szükséges közvetlenül a távolabbi sorkezelőhöz üzenetcsatornát nyitni, elég egy közbülső sorkezelőhöz, ami továbbítani fogja az üzeneteket a cél sorkezelőhöz.
A sorkezelőket fürtökbe is köthetjük könnyebb adminisztrálás, magasabb fokú rendelkezésre állás és terheléselosztás megvalósítása végett.
Egy speciális konfiguráció a kliens-szerver (vékony kliens) konfiguráció. Egy szerverhez több kliens is tartozhat. A kliens gépeken nem lesz sorkezelő, illetve azok objektumai, csak a szerveren, ami üzenetküldő szolgáltatásokat nyújt a kliens gépeknek, pontosabban az azokon futó alkalmazásoknak. Ennek a megoldásnak az előnye, hogy egy kliens gép könnyebben konfigurálható, és minimális erőforrást igényel, viszont a kliens és szerver közötti hálózatnak biztosítottnak kell lennie, ugyanis ez nem védett a kommunikációs csatorna megszakadásától. A kliens egy dedikált csatornán (MQI channel) keresztül csatlakozik a szerverhez. Ennek a csatornának kell folyamatosan rendelkezésre állnia. A kliens és szerver ugyanazon gépen is lehet. A szerveren elhelyezkedő sorkezelő nem csak a klienseket szolgálhatja ki, hanem ezen a gépen lévő alkalmazásokat is. Windows 2000 hálózaton támogatja az Active Directory-n keresztüli dinamikus kötést.

Programozás

A következő fejezet az MQ programozásáról fog bővebb információt nyújtani, bemutatva ezzel számos fogalmat, alkalmazástervezési mintát, megvalósítási technikát.

Támogatott nyelvek

Az MQ üzenetküldési lehetőségeit mind eljárás-, mind objektumorientált nyelvekben ki lehet használni. Ezek a következők: Assembler (OS/390), C, C++, COBOL, Java, PL/I, RPG (AS/400), TAL (Tandem NonStop Kernel), Visual Basic (Windows és Windows NT), valamint a Windows NT és Windows 2000 ActiveX és COM technológiája is támogatott.
A támogatott fordítók listáját az MQ Application Programming Guide tartalmazza.

Támogatott API-k

A programozáskor az egyik lehetséges API a *Message Queue Interface* (MQI), mely függvények és konstansok összessége, mellyel ki lehet használni az MQ összes lehetőségét.
Alternatív megoldásként az *Application Message Interface*-t (AMI) is lehet használni, amely egyszerűbb, magasabb absztrakciós szinten lévő felületet nyújt. Ekkor nem szükséges megérteni az MQI hívások részleteit, viszont nem is férhetünk hozzá az MQ specifikus funkcionalitásokhoz. Az AMI-t az IBM specifikálta és fejlesztette, és átvette az Open Application Group Inc. Az AMI használatával előírásokat készíthetünk bizonyos tulajdonságok összeállításával, pl. sor neve, prioritás, lejárati idő, és ezeket rendelheti hozzá az alkalmazás az üzenetekhez. Megkönnyíti a programozást, adminisztrálást, vizuális eszközöket biztosít az előírások összeállításához.
Java programozóknak lehet hasznos a *Java Message Service* (JMS), melyet a Sun Microsystem Inc. fejlesztett ki néhány társcéggel. A JMS is egy API, melyet a legtöbb üzenetalapú middleware implementáció támogat. Megalkotásának célja egy egységes, kellően egyszerű felület kialakítása Java alkalmazások számára, melyen keresztül az üzenetkezelés

minden fontosabb művelete elérhető legyen, mégis az ezt használó programok hordozhatóak legyenek. A JMS asszinkron üzenetalapú kommunikációt biztosít, és garantálja a pontosan egyszeri üzenetküldést. A JMS specifikáció először 1998. augusztusában jelent meg, a cikk írásakor legutóbbi verzió az 1.0.2b, mely 2001. augusztusában született. Az MQ 5.2 verzióig a Java támogatás külön komponens volt, a Java osztályokat (MQSeries classes for Java, MQSeries classes for Java Message Service) IBM honlapról lehetett ingyenesen letölteni, míg az MQ 5.3-ban a Java támogatás a termék szerves részét képzi.
A Common Messaging Interface (CMI) interfész az üzenetek dinamikus konstruálására és feldolgozására használható. Az üzeneteket reprezentációjuk ismerete nélkül lehet lekérdezni és módosítani. Támogatja mind az önleíró, programnyelv független struktúrákat (pl. XML), mind a programnyelv függő adattípusokat (Java és C nyelven).

Főbb tulajdonságok

Egy üzenet sorakoztató middleware-t használó alkalmazás főbb tulajdonságai a két program közötti kapcsolatmentes modellből adódóan a következők:

- a programok közt nincs direkt kapcsolat;
- a kommunikáció idő független;
- a munka kis, különálló programokkal megoldható;
- esemény alapú kommunikáció, szemben az eljáráshívással;
- üzenetek prioritása meghatározható, jelezve az üzenetek sorrendjét;
- biztonságosság;
- adatintegritás fontossága (tranzakció kezelés);
- visszaállíthatóság (minden adat perzisztensen tárolva van, log fájlok).

Ezen alkalmazások előnyei:

- kis, újra felhasználható alkalmazások;
- nincs szükség kommunikációs protokoll kialakítására, hálózati protokoll megismerésére;
- a címzett alkalmazásoknak nem kell állandóan futniuk;
- alkalmazások közti üzenetváltás platform független.

Tervezés

Üzenet sorakoztató middleware implementációkra épülő alkalmazások fejlesztésekor jelentős számú tervezési kérdés merül fel, és ezeket megválaszolva körvonalazódhat az alkalmazás felépítése, működése. Ezekből a kérdésekből szeretném a tipikusabbakat feltenni:

- Az üzenetek tervezésekor el kell dönteni, hogy az üzenet egyszeri típusú legyen-e, vagy várunk-e rá választ. Ha választ várunk, meg kell mondanunk az üzenetleíróba, hogy melyik sorba. El kell dönteni, hogy az üzenet és a rá adott válasz szinkronban legyen-e. Ha e mellett döntünk, meg kell adni egy lejárati időt (timeout), mely egy időtartam, amennyi időn belül ha nem érkezik válasz, az alkalmazás hibát észlel. Tranzakció kezelésnél meg kell gondolni, hogy mely üzeneteket kell egy munkaegységbe tenni. Prioritások használata szükséges-e, vagy elég az időrendi sorrend? Akarunk-e bizonyos üzeneteket külön azonosítani? Ha igen, akkor van lehetőség az üzenet azonosítóját magunk beállítani, illetve egy üzenetet az azonosítója alapján kivenni. A sorkezelő újraindítása után visszaállítódjanak-e a sorban lévő üzenetek? A visszaállítás nem perzisztens üzenetekre és dinamikus sorokra nem lehetséges. Az üzenetben legyen információ a feladóról? (Alapesetben a sorkezelő beleteszi a felhasználó azonosítóját (user id), de ezt az alkalmazásból felülírhatjuk.)

- Milyen típusú sorokat kell használni? Már meglévő sort kell használni, vagy az alkalmazásnak kell létrehoznia új sort? Használat után törölni kell-e a sort? Kell-e használni alias sorokat?
- Kell-e alkalmazásokat automatikusan indítani (triggering-re van-e szükség)? Vagy csak különleges üzenet esetén? Ha a feldolgozás nem elég gyors, kell-e újabb példányt indítani az alkalmazásból?
- Kell-e használni a vékony kliens megoldást?
- Mennyire kell védeni az adatokat, és fenntartani az integritást? Szükség van-e tranzakció kezelésre, külső tranzakció kezelőkkel kapcsolatban lesz-e? Perzisztensen kell-e mindent tárolni?
- Hibakezelés hogy legyen megoldva?

MQ technikák

A következő fejezetek a programozás szempontjából veszi át a jelentősebb MQ technikákat, melyeket használhatunk egy alkalmazás fejlesztésekor.

Üzenet várás

A következő technikák léteznek az üzenet várására:

- Megadott időközönként a sor megvizsgálása, jött-e új üzenet (*polling*).
- Az alkalmazás vár addig, míg egy üzenet megérkezik, vagy letelik egy időintervallum (*timeout*).
- Jel (*signal*) küldése, ha egy üzenet megérkezett.

Összekapcsolt válasz

Válasz küldése esetén a válasz üzenet üzenetleírójában a *correlation identifier* (`CorrelId`) mezőjét be kell állítani a kapott üzenet azonosítójára, így azonosítható, hogy melyik üzenetre jött a válasz.

Környezeti információ

Az üzenetleíró környezeti információ (*context information)* mezőjében állíthatjuk be, hogy ki rakta a sorba az adott üzenetet. Ez fontos lehet a biztonság, azonosítás, account kezelés, hibakeresés és útkeresés szempontjából is. Persze megadhatjuk azt is, hogy a sorkezelő egy alapértelmezett értékkel töltse ki ezt a mezőt. A környezeti információ két részből áll, az *identity context*-ből és az *origin context*-ből. Az előbbi tartalmaz információkat arról az alkalmazásról, mely először tette az üzenetet a sorba, az utóbbi arról az alkalmazásról, mely abba a sorba tette az üzenetet, amelybe éppen tartózkodik. Általában ezeket a sorkezelő állítja be, de egy alkalmazás megfelelő jogosultságokkal maga is megteheti.

Triggering

A *triggering* feltételektől függő automatikus programindítás. Ezek a feltételek a következők lehetnek:

- üzenet érkezik a sorba (EVERY);
- első üzenet érkezésekor (FIRST);
- ha a sorban lévő üzenetek száma elér egy határértéket (DEPTH).

A trigger üzenetek nem perzisztensek, csökkentve ezáltal a naplózást, növelve a teljesítményt. Ha egy sorra a triggering engedélyezve van, lehetőség van *process-definition list* objektum hozzákapcsolására ehhez a sorhoz. Ez leírja az alkalmazást, amelyik feldolgozza azt az üzenetet, mely a trigger üzenetet kiváltotta. A sorkezelő ezt az információt automatikusan

hozzáadja a trigger üzenethez. Minden sornak lehet külön ilyen objektuma, de akár egyet is megoszthatnak. Az indított alkalmazásnak és a trigger monitornak ugyanazon rendszeren kell futnia.
A triggering magába foglalja a következőket:

- Alkalmazás sor (*application queue*): az a sor, melyre ha teljesül a feltétel, akkor a sorkezelőnek trigger üzenetet kell generálnia.
- Process definition: a sorhoz van rendelve, információkat tartalmaz az alkalmazásról, aki ki fogja venni az üzenetet az alkalmazás sorból.
- Átviteli sor: trigger-starting channel esetén.
- Trigger esemény: akkor keletkezik, ha a feltétel teljesül, a sorkezelő kapja meg, ami ezután trigger üzenetet tesz az initiation queue-ba.
- Trigger üzenet: sorkezelő teszi az initiation queue-ba, az indítandó alkalmazásról nyújt információkat, amit a process definition-ből vesz. Formátuma meghatározott.
- Initiation queue: lokális sor, melybe a trigger üzenet kerül. Egy sorkezelőben több ilyen is lehet, különböző alkalmazás sorokhoz rendelve.
- Trigger monitor: állandóan futó program, mely az initiation queue-(ka)t figyeli, és az üzenet alapján elindítja az alkalmazást, mely ki fogja venni az üzenetet az alkalmazás sorból. A trigger monitor alkalmazást külön kell elindítani.

A trigger esemény kiváltását finoman lehet szabályozni bizonyos feltételekkel.

MQ jelentések (report)

A jelentés üzenetet egy üzenet sorba tételekor lehet kérni. A jelentés üzeneteket generálhatják sorkezelők, üzenetcsatorna ágensek és alkalmazások is. Ezek akkor is keletkezhetnek, ha az alkalmazás nem számít rá. A következő jelentések kérhetők egy alkalmazásban:

- *kivétel* (exception): olyan üzenetekre válasz, melynek az *exception flag*-je be van állítva (generálhatja üzenetcsatorna ágens vagy alkalmazás);
- *lejárati* (expiry): egy alkalmazás olyan üzenetet akar elérni, melynek a lejárati ideje lejárt (sorkezelő küldheti);
- megérkezési (confirm-on-arrival - COA): jelzi, hogy egy üzenet elért egy sort (a sorkezelő generálhatja);
- *kézbesítési* (confirm-on-delivery - COD): jelzi, hogy egy alkalmazás kivette az üzenetet a sorból (a sorkezelő generálhatja);
- *sikeres akció* (positive action notification - PAN): jelzi, hogy az üzenetben kért akció végre lett hajtva (alkalmazás generálhatja);
- *sikertelen akció* (negative action notification - NAN): jelzi, hogy az üzenetben kért akció nem lett sikeresen végrehajtva (alkalmazás generálhatja).

Egy üzenet sorba tételekor többfajta jelentés üzenetet is kérhetünk.
A jelentés üzenetek a következők egyikét tartalmazhatják:

- a teljes eredeti üzenet;
- az eredeti üzenet első száz bájtja;
- nem tartalmaz semmit az eredeti üzenetből.

A jelentés üzeneteknek megadhatók a `MsgId` és `CorrelId` mezőire ugyanazok a feltételek, mint a válasz üzenet esetében.
Lehetőség van arra, ha egy kivétel üzenet kiváltódik, akkor az eredeti üzenetet eldobja a sorkezelő. Ez erőforrás megtakarítással jár, hiszen ekkor nem kerül a dead-letter queue-ba.
Meg kell adni, hogy a jelentés üzenetek melyik sorkezelő melyik sorába érkezzenek.
Ha a jelentés üzenetek nem kézbesíthetőek, akkor a dead-letter queue-ba kerülnek.

Az 5-ös verzióban jelentek meg a szegmentált üzenetek, azaz egy üzenet feldarabolva, több üzenetként jut el a cél sorba. Ezeknek a használatakor a jelentés üzenetek kezelése is jelentősen módosul, rengeteg beállítást biztosítva.

Fürtök és üzenet összetartozások

Az alkalmazás logikája is tisztább, ha az alkalmazás számára a fürtözés transzparens, növelve a rendelkezésre állást és a skálázhatóságot.

MQI API

Az MQI (Message Queue Interface) tartalmaz *hívásokat* (call), mellyel el lehet érni az alkalmazásból a sorkezelőket és azok szolgáltatásait, *adatstruktúrákat* (structure), melyeket átad, illetve megkap a sorkezelőktől, illetve *elemi adattípusokat* (elementary data type) szintén az adatcserére. Ezen kívül platformtól, fordítótól függően tartalmazhat fejléc fájlokat, könyvtárakat, példaprogramokat.

Műveletek

A következő műveleteket lehet elvégezni az alkalmazásból az MQI hívásokkal:

- csatlakozás és lecsatlakozás a sorkezelőről (`MQCONN`, `MQCONNX`, `MQDISC`);
- objektum megnyitása és lezárása (`MQOPEN`, `MQCLOSE`);
- üzenet sorba helyezése (`MQPUT`, `MQPUT1`);
- üzenet olvasása (sorban marad) és kivétele (`MQGET`);
- objektumok tulajdonságainak lekérdezése (`MQINQ`);
- objektumok tulajdonságainak beállítása (`MQSET`);
- munkaegység elkezdése, jóváhagyása és visszagörgetése (`MQBEGIN`, `MQCMIT`, `MQBACK`), ezekről a tranzakció-kezelésről szóló részben lesz szó;
- sorkezelők frissítésének (update) irányítása.

Elemi adattípusok

C nyelvhez az MQ a következő elemi adattípusokat vagy struktúrálatlan mezőket nyújtja (a táblázat tartalmazza a megfelelő Visual Basic adattípusokat is):

C adattípus	Visual Basic adattípus	Leírás
`MQBYTE`	String*1	Egy bájt adat.
`MQBYTEn`	String*n	16, 24, 32 vagy 64 hosszú bájttömb.
`MQCHAR`	String*1	Egy bájton ábrázolt karakter.
`MQCHARn`	String*n	4, 8, 12, 16, 20, 28, 32, 48, 64, 128, 256 hosszú karaktertömb.
`MQHCONN`	Long	Kapcsolatkezelő (32 bites).
`MQHOBJ`	Long	Objektumkezelő (32 bites).
`MQLONG`	Long	32 bites előjeles egész.
`PMQLONG`	Nincs megfelelő adattípus	`MQLONG` típusú mutató.

Az MQI hívások visszatérési értékeiről a hibakezelésről szóló részben lesz szó.
Az MQI struktúrákat (mezők csoportja) biztosít a hívások paramétereihez és visszatérési értékeihez, nevesített konstansokat, illetve eljárásokat vagy függvényeket minden támogatott programozási nyelvhez (lásd feljebb), illetve struktúrák mezőihez alapértelmezett értékek tartozhatnak.

Itt kell megemlíteni a distribution list-et, amely egy lehetőség, hogy egy üzenet több sorba is lehessen elküldeni az MQPUT és MQPUT1 hívásokkal. Az a publish and subscribe elterjedésével a háttérbe szorul.
Az MQCONN és MQCONNX hívásokkal megadott vagy alapértelmezett sorkezelőhöz lehet kapcsolódni. (A default sorkezelő nevét Windows-ban a registry tartalmazza.) A sorkezelőnek lokálisnak kell lennie, azaz ugyanazon a rendszeren kell lennie, mint az alkalmazás. Az MQCONN hatásköre az a szál, ahonnan a hívás indult, azaz a visszaadott kapcsolatkezelő csak ebben a szálban érvényes. Másik szálban való felhasználás esetén a hívások hibát adnak vissza, hiszen ebben a szálban érvénytelen. Ha mégis szükségünk van a kapcsolatkezelőre, akkor a másik szálból is MQCONN hívást kell kezdeményeznünk. A különböző szálakban különböző sorkezelőkhöz kapcsolódhatunk. Az MQCONNX hívás annyiban tér el az MQCONN hívástól, hogy meghatározhatjuk a csatlakozás módját. Ez lehet ugyanaz, mint az MQCONN esetében, azaz az alkalmazás és a lokális sorkezelő ágens különböző folyamathoz (process) tartoznak (MQCNO_STANDARD_BINDING paraméterrel), illetve *trusted* alkalmazás esetén ugyanahhoz a folyamathoz tartoznak (MQCNO_FASTPATH_BINDING paraméterrel). az utóbbi esetén az alkalmazás meghibásodás esetén magával ránthatja a sorkezelőt is. Az MQDISC hívás szünteti meg a kapcsolatot a sorkezelővel, paraméterként a kapcsolatkezelőt kell megadni. Ezzel a kapcsolatkezelővel ezután nem végezhetünk műveletet, mert hibát kapunk. Az MQDISC automatikusan lezár minden nyitott objektumot is (implicit MQCLOSE).
Ahhoz, hogy üzenetet tegyünk egy sorba, vagy vegyünk ki onnan, objektum tulajdonságát kérdezzük le, vagy állítsuk be, az objektumot meg kell nyitni. Egyetlen kivétel, mikor csak egy üzenetet akarunk a sorba tenni, ilyenkor használjuk az MPUT1 hívást. Ekkor nem kötelező az MQOPEN használata. MQOPEN használata előtt kapcsolódni kell egy sorkezelőhöz. A következő négy objektum nyitható meg: sor, névlista, folyamat definíció és sorkezelő. Egy objektumra többször is használható az MQOPEN, ekkor mindig más kapcsolatkezelőt ad vissza (, így egy sornál például az egyikkel kivehetünk, a másikkal betehetünk üzenetet), illetve egy hívással egyszerre több objektumot is megnyithatunk. Az MQOPEN hívásnál meg kell adni az objektum leírót (MQOD), ami meghatározza, hogy melyik objektumot nyissa meg. Ezt a sorkezelő módosíthatja a megnyitáskor. Ekkor névfeloldás is történik, ha távoli vagy alias sorra hivatkozunk, azaz meghatározódik a lokális sorkezelő és sor neve. Az MQOPEN hatásköre az a szál, ahonnan a hívás indult.
Az MQOPEN hívásnál adhatunk meg különböző paramétereket a megnyitás módjának pontosítására. Ezek a következők lehetnek:

- Megadható, hogy a sorba tett összes üzenet azonos sor példányhoz fusson be (fürtöknél). Paraméterek: MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED.
- A sort csak üzenet belerakására nyitunk meg. Paraméter: MQOO_OUTPUT.
- A sort csak üzenetek böngészésére nyitunk meg (távoli sornál és distribution list esetén nem lehetséges). Paraméter: MQOO_BROWSE.
- A sort csak üzenet kivételére nyitunk meg. Paraméter: MQOO_INPUT_EXCLUSIVE (kizárólagos hozzáférés: csak a megnyitó alkalmazás férhet hozzá), MQOO_INPUT_SHARED (elosztott hozzáférés: más alkalmazás is hozzáférhet).
- Objektum tulajdonságainak beállítása, kiolvasása. Paraméterek: MQOO_SET, MQOO_INQUIRE.
- Az üzenetkörnyezet beállítására is adhatunk meg paramétereket, melyekkel megkülönböztethetjük, hogy az üzenetet a sorba rakó alkalmazásról vagy felhasználóról legyen információ az üzenetkörnyezetben.
- Megadhatunk felhasználó azonosítókat, akik szintén hozzáférhetnek az objektumhoz, ugyanis az MQOPEN használatánál azonosítás történik, és a felhasználónak

rendelkeznie kell jogokkal, hogy ezt megtegye. Paraméter: MQOO_ALTERNATE_USER_AUTHORITY.

- Csendes módnál megadhatjuk, hogy a megnyitás okozzon-e hibát. Alapesetben nem. Paraméter: MQOO_FAIL_IF_QUIESCING. (A csendes módról a hibakezelés részben írok többet.)

Az MQOPEN hívással hozhatunk létre dinamikus sorokat is, ekkor meg kell adni a modell sort, mely alapján a dinamikus sor létrejön. Bezárás után a sor és az összes benne lévő üzenet törlődik.
Az MQCLOSE használata mindenképp ajánlott, az objektumok bezárása jó programozási gyakorlat. Az objektum bezárása független a szinkronizációs pontoktól, ezért tetszőleges sorrendben követhetik egymást.
Az üzenet sorba rakásához az MQPUT vagy MQPUT1 hívást kell használnunk. Az utóbbit használjuk, ha csak egy üzenetet szeretnénk a sorba tenni, mert ez a hívás megnyitja a sort, beletesz egy üzenetet, majd lezárja a sort. A következő paramétereket kell megadni (zárójelben a típusa):

- kapcsolatkezelő (HCONN), melyet az MQCONN vagy MQCONNX ad vissza;
- sor objektumkezelője (HOBJ), melyet az MQOPEN ad vissza MQPUT esetén, vagy objektum leíró (MQOD) struktúrát, mely a sort azonosítja MQPUT1 esetén;
- üzenetleíró (MQMD);
- vezérlő információk üzenet sorba tételéhez (put-message option - MQPMO);
- az üzenet hossza (MQLONG);
- az üzenet tartalmát.

A sorkezelő módosítja az üzenetleíró és a vezérlő információkat hordozó struktúra tartalmát a betett üzenetnek megfelelően, így ha ezzel újra üzenetet akarunk a sorba helyezni, alapállapotba kell állítani. Így ezek a struktúrák input/output paraméterként viselkednek. Az üzenetleíró mezőiről már szó esett, így nem említem meg itt külön. Az MQPMO a következő mezőket tartalmazza:

- StrucId: négy karakteres mező, a vezérlő információk struktúrát azonosítja, mindig PQPMO_STRUC_ID-t kell értékül adni neki.
- Version: a struktúra verziószámát adja meg, alapesetben PQPMO_VERSION_1, distribution list használatakor PQPMO_VERSION_2. MQPMO_CURRENCT_VERSION esetén a legújabbat állítja be.
- Options: ezekkel tudjuk megadni az üzenet sorba tételének paramétereit. Megadható, hogy az üzenet munkaegységbe tartozik-e, mennyi környezeti információ tartozzon az üzenethez, és mi alapján kell beállítani, csöndes módban hibát adjon-e vissza a művelet, a csoportosítás vagy szegmentálás megengedett-e, az új üzenet azonosítójának és korrelációs azonosítójának generálása mi alapján történjen, és az üzenetek és szeletek sorba tételének sorrendje.
- Context: ez tartalmazza a nevét a sor objektumkezelőjének, melyből a környezeti információ másolva legyen, ha az Options mezőben ezt adtuk meg.
- ResolvedQName: a cél sor feloldott nevét tartalmazza.
- ResolvedQMgrName: a cél sorkezelő feloldott nevét tartalmazza.
- Abban az esetben, ha distribution list-et használunk, és a verziószáma PQPMO_VERSION_2, akkor további mezők is szerepelnek a vezérlő információkban, úgymint RecsPresent, PutMsgRecFields, PutMsgRecOffset, PutMsgRecPtr, ResponseRecOffset, ResponseRecPtr, melyek jelentésére most nem térnék ki.

Az adatokat egy pufferbe kell megadni, mely tartalma bármi lehet, és az MQPUT hívásnak ennek a puffernek a címét kell megadni. Az üzenet maximális mérete a sorkezelőnél és a sornál megadott maximális értékek minimuma. Az üzenet méretéhez hozzá kell adni az üzenet fejlécek méretét, melyek az üzenethez kapcsolódhatnak. Ezek az átviteli fejléc (transmission header), dead-letter fejléc (dead-letter header - MQDLH), distribution list fejléc (distribution list header - MQDH).
Az MQGET hívással lehet kivenni az üzenetet a sorból, illetve úgy hozzáférni, hogy az a sorban maradjon. Ennél a hívásnál a következő paramétereket kell megadni:

- kapcsolatkezelő (HCONN);
- sor objektumkezelője (HOBJ);
- üzenetleíró (MQMD);
- vezérlőinformációk üzenet kivételéhez (MQGMO);
- a puffer mérete, melybe az üzenet tartalmát kell másolni (MQLONG);
- a puffer címe.

Ha megvan, hogy mely üzenetet akarjuk kivenni, akkor pontosan állítsuk be az MQMD mezőt, de ha nem, akkor a hívás automatikusan az első üzenetet fogja visszaadni. Azt, hogy melyik az első a prioritás, a sor MsgDeliverySequence tulajdonsága és a vezérlőinformáció MQGMO_LOGICAL_ORDER opciója határozza meg.
A vezérlőinformációk struktúra a következő mezőket tartalmazza:

- StrucId: négy karakteres mező, a vezérlő információk struktúrát azonosítja, mindig PQGMO_STRUC_ID-t kell értékül adni neki.
- Version: a struktúra verziószámát adja meg, alapesetben PQGMO_VERSION_1, distribution list használatakor vagy az üzenetek logikai sorrendjében való kivételkor PQGMO_VERSION_2. MQGMO_CURRENCT_VERSION esetén a legújabbat állítja be.
- Option: az üzenet kivételének opcióit adhatjuk meg. Beállíthatjuk, hogy várunk-e az üzenetre, vagy egyből visszatérjen a hívás, ha nincs aktuálisan üzenet. Beállítható, hogy az üzenet kivétele munkaegységbe tartozzon-e; nem perzisztens üzenet szinkronizációs ponton kívül essen-e; az üzenet a sorban maradjon-e; az üzenet kiválasztása kritérium vagy a kurzor szerint történjen-e; a hívás sikeres legyen-e, ha az üzenet hosszabb, mint a megadott puffer mérete; a hívás hibát adjon-e vissza csöndes mód esetén; adatkonverzió történjen-e; csoportokra és szegmensekre történő paramétereket.
- WaitInterval: ezredmásodpercekben kell megadni, hogy a hívás mennyi ideig várjon maximum, ha nincs üzenet.
- Signal1: a jel típusát kell megadni, melyet az alkalmazás kap, ha megérkezik az üzenet.
- Signal2: a jel azonosítóját kell megadni.
- ResolvedQName: a cél sor feloldott nevét tartalmazza.
- ResolvedQMgrName: a cél sorkezelő feloldott nevét tartalmazza.
- MatchOptions: a kiválasztási kritériumot vezérli.
- GroupStatus: az üzenet egy csoport tagja-e.
- SegmentStatus: egy logikai üzenet egy szelete-e az üzenet.
- MsgToken: csak OS/390-en, egyértelműen azonosít egy üzenetet.
- ReturnedLength: az üzenet méretét adja meg.

A puffer méretét három módon határozhatjuk meg. Vagy tudjuk, hogy a küldő alkalmazás mekkora üzeneteket fog küldeni; vagy meghatározunk egy méretet, és ha nagyobb, akkor megadjuk, hogy a kivétel hibát jelezzen nagyobb üzenet esetén, és a puffer méretet

nagyobbra állítjuk, és újra megpróbáljuk kivenni az üzenetet; vagy a sorkezelő és sor által meghatározott legnagyobb méretet adjuk meg.
Az üzeneteket kivehetjük prioritás szerint, kivehetjük fizikai sorrend szerint (, azaz a sorba kerülés sorrendjében), vagy csoportoknál és szegmentálásnál megadott logikai sorrend szerint.
Abban az esetben, ha a sor nem tartalmaz üzenetet, vagy nem azt, amire szükségünk van, megadhatjuk, hogy várunk az üzenetre, és ekkor meg kell adni egy időintervallumot is, amíg maximum várunk az üzenetre. Másik megoldás, hogy jelzést (signaling) alkalmazunk, amikor a program futása nem blokkolódik, mint az első esetben, hanem az üzenet megérkezésekor a Signal1 mezőben megadott ablak kap egy jelet, hogy megérkezett az üzenet. Ebben az esetben az alkalmazás szála futhat tovább, és az operációs rendszerre bízzuk az alkalmazás jelzését.
Abban az esetben, ha a sort böngészésre (üzenetek olvasására, azok kivétele nélkül) nyitjuk meg, létrejön egy kurzor (browse cursor), mely egy logikai mutató egy üzenetre. Ha meghívjuk az `MQGET` hívást, meg kell adnunk, hogy az első üzenetet vesszük-e ki (`MQGMO_BROWSE_FIRST`), vagy a következőt (`MQGMO_BROWSE_NEXT`).
Azt az üzenetet, melyre a kurzor mutat, el tudjuk távolítani a sorból, az `MQGET` hívást kell újra hívnunk, de az `MQGMO` Options mezőjébe be kell állítanunk az `MQGMO_MSG_UNDER_CURSOR` értéket.
Az objektumok tulajdonságainak lekérésére az `MQINQ` és `MQGET` hívásokat használjuk. Bizonyos tulajdonságokat csak az objektum definiálásakor adhatunk meg. Nem minden tulajdonságot lehet `MQSET` hívással beállítani, helyette a MQSC parancsokat kell használni.
Mind a lekérés mind a beállítás előtt meg kell nyitni az objektumot.
Az `MQINQ` hívásnak a következő paramétereket kell megadni:

- kapcsolatkezelő (`HCONN`);
- objektumkezelő (`HOBJ`);
- szelektorok számát;
- szelektorok tömbjét, melyeket `MQCA_*` (karakteres) vagy `MQIA_*` (numerikus vagy konstans) prefix-szel kezdődő konstansokkal adhatunk meg, melyek a tulajdonságot azonosítják;
- az egész értékű tulajdonságok száma;
- a karakteres tulajdonságok értékét tartalmazni fogó puffer méretét.

Beállítani csak a következő tulajdonságokat lehet: InhibitGet, DistList, InhibitPut, TriggerControl, TriggerType, TriggerDepth, TriggerMsgPriority, TriggerData. Paraméterei megegyeznek az előző MQI hívás paramétereivel.

Data-conversion exit

Az üzenet leírót az alkalmazás hozza létre, mikor az üzenetet beteszi a sorba. Ennek konverziója szükséges a platformoktól függően, ugyanis az MQ-nak meg kell értenie az üzenet leírót.
Az alkalmazás adat nem konvertálódik automatikusan. Ha az üzenet olyan platformok között mozog, melyen eltér a `CodedCharSetId` és `Encoding` mező, a konverziót végezheti a sorkezelő beépített rutinokkal (, abban az esetben, ha az alkalmazás adat beépített formátumot használ, pl. `MQFMT_STRING`), vagy egy program exit, más néven *data-conversion exit* (, ha nem beépített formátumú az adat).
A konvertálásra két helyen is sor kerülhet. Egyrészt üzenet átküldésekor a forrás rendszeren a csatorna konvertálhat, ilyenkor a forrás rendszeren kell az exit-et elhelyezni, illetve az `MQGET`

hívásnál is történhet a konverzió, így megakadályozhatjuk, hogy az üzenet több csomóponton áthaladva mindig konvertálódjon.
A data-conversion exit-ben csak egy MQI hívás használható, az `MQXCNVC` hívás, mely karaktereket konvertál az egyik karakterhalmazból a másikba. Csak data-conversion exit-ben használható.
A data-conversion exit írásához az MQ ad egy váz forrásprogramot (skeleton source code), a fentebb említett `MQXCNVC` hívást, és egy segédprogramot, mely a C-ben deklarált saját adattípus alapján elkészíti egy kódrészletet, mely a konverziót végzi. Ezt a kódrészletet kell majd beilleszteni a vázba.

Objektumorientáltság

Az MQI hívások használata helyett a MQ egy másik módot is biztosít az objektumorientált nyelvek számára, melynek neve *MQSeries Object Model*. Hívások, struktúrák és elemi adattípusok helyett osztályokat biztosít az MQ funkcióinak elérésére, illeszkedve ezáltal az OO módszertanba. A hívásokat így módszerek, struktúrákat az osztályok tulajdonságai reprezentálják. Ezekre való hivatkozások megegyeznek az `MQINQ` és `MQSET` hívásokkal.
Az MQSeries Object Model a következő osztályokat nyújtja:

- `MQQueueManager` reprezentálja a sorkezelőt, módszerei a `Connect()`, `Disconnect()`, `Commit()` és `Backout()`. A tulajdonságokra hivatkozás automatikusan kapcsolódik a sorkezelőhöz, a példány megsemmisítése pedig automatikusan lecsatlakoztat.
- `MQMessage` reprezentálja az üzenetet, mely magában foglal egy puffert. Ez tartalmazza mind az üzenetleírót, mind pedig az üzenet tartalmát. Tulajdonságokat tartalmaz a leíró mezőinek megfelelően és módszereket a pufferbe való íráshoz és olvasáshoz.
- `MQPutMessageOptions` reprezentálja az MQPMO struktúrát.
- `MQGetMessageOptions` reprezentálja az MQGMO struktúrát.
- `MQProcess` reprezentálja a triggering-hez használatos folyamat definíciókat.
- `MQDistributionList` reprezentálja a distribution list-et, melyből példányosított objektumok elemei az MQDistributionItem objektumok.
- `MQDistributionItem` reprezentál egy célt a distribution list-ben. Reprezentálja az `MQOR`, `MQRR` és `MQPRM` struktúrákat.

Az MQSeries Object Model használatakor a módszereknek nem a kapcsolat- és objektumkezelőket kell átadni, hanem a megfelelő objektum referenciákat. A visszatérési értékek is az objektum tulajdonságain keresztül érhetőek el.
Az MQSeries Object Model a C++, Java, LotusScript (MQLSX) és ActiveX (MQAX) nyelvekben illetve technológiákban van megvalósítva.

Tippek a teljesítmény növeléséhez

A feldolgozás legyen párhuzamos a felhasználó gondolkodási idejével, azaz pl. előbb tegyük ki a bevitelt váró panelt, közben inicializálódhat az alkalmazás. A különböző szerverekről az adatok gyűjtése legyen párhuzamos.
A kapcsolatok és sorok maradjanak nyitva, és használjuk fel újra ahelyett, hogy lezárnánk őket, majd újra megnyitnánk.
Ha csak egy üzenetet kell a sorba tenni, használjuk az `MPUT1` MQI hívást.
Üzeneteket érdemes munkaegységekbe szervezni.
A visszaállítást nem feltétlenül megkövetelő üzenetek ne legyenek perzisztensek.

Hibakezelés

Az alkalmazás készítésekor kétfajta hibára kell felkészülni. Amikor csak lehetséges, a sorkezelő már az MQI hívásnál hibát jelez, ezt *lokálisan felismert hibának* (locally determined error) nevezzük. Abban az esetben, ha távoli sorba próbálunk üzenetet küldeni, a hiba később is felmerülhet, erről a hibát felismerő sorkezelő fog üzenetet küldeni az eredeti programnak. Az ilyen hibát *távol felismert hibának* (remotely determined error) nevezzük.

Lokálisan felismert hiba

A lokálisan felismert hibának három fajtája van:

- MQI hívás közben bekövetkezett hiba: közvetlen jelzi az MQI hívás visszatérési értéke. Ezt két felé bonthatjuk, *elvégzési kódra* (completion code) és *okozati kódra* (reason code). Az előbbi jelezheti, hogy a művelet sikeresen (`MQCC_OK`), részben sikeresen (`MQCC_WARNING`) vagy egyáltalán nem sikerült (`MQCC_FAILED`). Az utóbbi hiba okát jelzi. Az alkalmazás készítésekor az összes lehetséges hibára fel kell készülni.
- Rendszer megszakítás: a rendszer azon része hibázik, melytől függ az alkalmazás. Rendszerhiba következtében történhet olyan, hogy vissza kell állítani egy komponenst, amitől függ az alkalmazás, például a sorkezelőt, melyhez kapcsolódik. Az alkalmazást úgy kell megtervezni, hogy ilyen hiba esetén se legyen adatvesztés. Ez függ az alkalmazott platformtól is, Windows rendszereken szinkronizációs pontokat deklaráljuk `MQCMIT` és `MQBACK` hívásokkal, biztosítva az adatok integritását. Azon üzeneteket, melyeket ilyen hiba esetén sem akarunk elveszteni, perzisztens sorba kell rakni, hogy visszaállíthatók legyenek.
  Abban az esetben, ha az operátor leállítja a sorkezelőt, alapesetben a *csendes állapotba* (quiescing state) kerül, azaz a programoknak amikor csak lehetséges, terminálniuk kell. A kicsi és gyors programok figyelmen kívül hagyhatják ezt, és befejeződhetnek úgy, ahogy általában szoktak, de a nagy, hosszú futásidejű, vagy üzenetre váró programoknak használni kell a *hiba csendes állapot esetén* (fail if quiescing) opciót, a `MQCONN`, `MQCONNX`, `MQPUT`, `MQPUT1` és `MQGET` hívásoknál, és azután marad idejük a tiszta befejezésre, tranzakciók jóváhagyására vagy visszagörgetésére.
  Abban az esetben, ha a sorkezelőt kényszerítik a leállásra, akkor `MQRC_CONNECTION_BROKEN` okozati kódot adnak vissza az MQ hívások.
- Az üzenet hibás adatot tartalmazhat. Abban az esetben, ha munkaegységeket használunk, és az alkalmazás nem tudja megfelelően feldolgozni az üzenetet, az MQ visszagörgeti az MQGET hívást. Ebben az esetben egy számláló (az üzenet BackoutCount mezője) inkrementálódik, és az alkalmazásnak fel kell készülnie rá, hogy ha ez elér egy bizonyos értéket, akkor fel kell deríteni a hibát, hogy az üzenet miért hibás. A legtöbb platformon ez az érték megmarad a sorkezelő újraindítása után is.

Jelentés üzenetek felhasználása hibakeresésre

Távoli sorba küldött üzeneteknél nem mindig derül ki a hiba azonnal az MQI híváskor, de felhasználhatóak a jelentés üzenetek arra, hogy értesítsük a küldő alkalmazást, hogy a célalkalmazás sikeresen feldolgozta-e az üzenetet. Mind jelentés üzenetek küldésére, mind azok feldolgozására van lehetőség.
Jelentés üzenetek készítésekor figyelembe kell venni a kapott üzenet Report mezőjét, ami jelzi, hogy a küldő alkalmazás kíváncsi-e rá, hogy sikerült-e a célalkalmazásnak feldolgoznia az üzenetet. Ha szükség van a jelentés üzenetre, akkor meg kell vizsgálni, hogy az eredeti üzenet egésze, első 100bájtja, vagy semennyi ne kerüljön a jelentés üzenetbe, hogy mit kell

tenni az eredeti üzenettel (eldobni vagy a dead-letter sorba tenni), hogy az üzenet MsqId vagy CorrelId mezői szükségesek-e. A Reason mezőbe kell tenni a hiba okát. A jelentést a válasz sorba kell tenni. A küldő alkalmazásba be kell építeni a jelentés üzenetek feldolgozását.

Távol felismert hiba

Ha a sorkezelőkön áthaladva hiba történik, akkor az nem derül ki egyből az MQI hívásnál, hanem az üzenetben kell megadnunk, hogy a sorkezelő mit tegyen, ha nem tudja az üzenetet betenni a cél sorba. Három eset lehetséges: üzenet kézbesítésének újrapróbálása, visszaküldése a feladónak, illetve a dead-letter sorba helyesése. Az elsőnél meg kell adni, hogy milyen időközönként próbálkozzon, és maximum hányszor. Megadhatunk egy programot is, un. *retry exit program*-ot, melyet automatikusan elindít a sorkezelő. Az üzenet visszaküldésénél a jelentés üzeneteknél tárgyaltakat kell figyelembe venni. A kézbesítetlen üzenetek sorát a sorkezelő installálásakor kell megadni, és az alkalmazáskor kapcsolódáskor lekérhetjük a nevét. Abban az esetben, ha a sorkezelő ide helyez egy üzenetet, hozzácsatol egy fejlécet.

Tranzakció-kezelés

Egy munkaegységbe tartozó üzeneteket ugyan betesz a sorkezelő a sorba, de a többi alkalmazás számára csak akkor válnak láthatóvá, ha a munkaegységet végrehajtják, viszont visszagörgetés esetén a sorkezelő eltávolítja őket a sorból. Ugyanígy a kivételkor is csak akkor veszi ki az MQ az üzenetet ténylegesen a sorból, ha az üzenetet tartalmazó munkaegységet végre kell hajtani.
A synchpoint coordination az a folyamat, mely által a munkaegységek jóváhagyódnak vagy visszagörgetődnek. Üzenetek sorba rakásakor (`MQPUT, MQPUT1`) és kivételekor (`MQGET`) is meg kell adni, hogy az üzenet syncpoint control alatt van-e. Ekkor a `MQPMO_SYNCHPOINT` értéket kell megadni a `PQPMO` struktúra egy opciójának a sorba rakásnál, és `MQGMO_SYNCPOINT` értéket a `PQGMO` struktúránál a kivételnél. `MQDISC` automatikusan szinkronizációs pont is, a munkaegység jóváhagyódik. Az alkalmazás hibás leállása esetén visszagörgetés van.
Jóváhagyni az `MQCMIT`, visszagörgetni az `MQBACK` hívással lehet.
Lokális munkaegység esetén a sorkezelő végzi a szinkronizációs pont koordinációt, és egyfázisú végrehajtó protokollt használ. Ebben az esetben a munkaegység kezdetét az üzenetek sorba rakásakor vagy kivételkor kell megadni, és az `MQCMIT` vagy `MQBACK` hívással kell lezárni.
Globális munkaegység esetén más erőforrás kezelőhöz tartozó erőforrásokat is frissíteni kell, aminek irányítását végezheti maga a sorkezelő (*internal syncpoint coordination*), vagy más tranzakció-kezelő (*external synchpoint coordination*). A belső szinkronizációs pont irányításnál a globális munkaegységet az `MQBEGIN` hívással kell kezdeni és az `MQCMIT` vagy `MQBACK` hívással kell lezárni. Külső tranzakció-kezelő esetén az `MQBEGIN`, `MQCMIT` és `MQBACK` hívások nem használhatóak, hibát adnak vissza. Ebben az esetben a tranzakció kezelők egy interfészt nyújtanak az alkalmazás számára a tranzakciós szolgáltatások elérésére.

Vékony kliens

Az MQ kliens az MQ része, és feltelepíthető mind arra a gépre, amelyen az MQ fut (fejlesztési és tesztelési célokat szolgálandó), mind egy másik gépre. Egy MQ alkalmazás futhat az MQ kliensen (továbbiakban kliens) is, amely egy kommunikációs csatornán kommunikál az MQ-val (továbbiakban szerver).
Az alkalmazást annyiban kell módosítani, hogy a klienshez tartozó könyvtárakkal kell összelinkelni, kivéve signal és globális munkaegységek esetén.

A kliens és a szerver között állandó kapcsolatnak kell lennie a szinkron kommunikáció miatt. A kapcsolat az `MQCONN` vagy `MQCONNX` hívásokkal jön létre, és az `MQDISC` hívásig tart. A kommunikációs csatorna neve MQI csatorna (MQI channel).
A kliens használatának előnyei:

- Nincs szükség az egész MQ implementáció telepítésének a kliens gépen, így futhat pl. DOS, Windows 3.1 vagy Windows 9x platformokon is.
- A kliensnek sokkal kisebb az erőforrásigénye.
- A rendszeradminisztráció egyszerűsödik.
- Egy kliens egyszerre több eltérő platformon üzemelő sorkezelőhöz is kapcsolódhat.
- Más átviteli protokollt használó csatornák is használhatóak.

5.3 verzió újdonságai

Ez a fejezet leírja azokat az újdonságokat, melyek a WebSphere MQ 5.3 termékben jelentek meg. Az előző fejezetekben már utaltam rájuk, itt csak összegyűjtöm őket. Az előző verzió az 5.2-es verziószámmal rendelkezett, és annak a neve is IBM MQSeries volt, és az 5.3-as-tól kezdve keresztelték át, a négy fő termékvonal bevezetésével. Az új verzióba bekerült a WebSphere MQ classes for Java és WebSphere MQ classes for Java Message Service (JMS) is, melyek Java osztályokat biztosítanak az MQ funkcióinak elérésére, illetve implementálják a JMS API-t. Az előző verzióhoz ezeket külön kellett letölteni. Az új verzió olyan komponenseket tartalmaz, melyek MSCS támogatást nyújtanak, melyet az előző verzióhoz szintén le kellett tölteni SupportPac MC74 néven. A legnagyobb újítás, hogy a WebSphere MQ támogatja az Secure Socket Layer (SSL) protokollt. Az új verzióból több jellemző is el lett távolítva, ilyen a teljes dokumentáció, melyet külön kell letölteni; a Lotus Script Extension, ehelyett Java nyelven lehet elérni a Notes-ot; a Web Administration Server és az Internet Gateway.

Konklúzió

Dolgozatomban első részében általános leírást adtam az elosztott rendszerekről, azok kialakulásáról. Látható, hogy ilyen rendszerekre egyre nagyobb az igény, a különböző méretű hálózatok elterjedésével. A programozóknak ugyanazon problémákkal kell megküzdeniük, ezért alakultak ki a middleware-ek, amelyek megoldásokat, API-kat nyújtanak, elrejtik a platformspecifikus részeket. Sajnos a fogalom jelentése nem egységes, ezért próbáltam valamilyen definíciót adni, illetve csoportosítottam funkcionalitásuk, felépítésük szerint.
A middleware-ek megbízhatóságuk növekedése, egyre több funkció biztosítása miatt beszivárogtak az üzleti megvalósítások közé is, így biztosítaniuk kellett olyan követelményeket, mint a tranzakció kezelés, többszintű biztonság megvalósítása. A harmadik és negyedik fejezetben ezeket részleteztem.
Az IBM Websphere MQ egy kitűnő megvalósítása az üzenet sorakoztató middleware-eknek, minden funkciót biztosít, ami egy ilyen terméktől elvárható. Ezen megvalósítás részleteinek leírása nem csak a termék megismerése szempontjából fontos, hanem egyfajta összehasonítási alapot is nyújt, illetve bevezet egy üzenetközpontú elosztott alkalmazás fejlesztésébe, illetve már kész alkalmazások integrációjának megvalósításába.
Dolgozatom egyik fő célja rámutatni, hogy az ilyen jellegű alkalmazásoknál mennyire fontos a szabványok betartása, implementálása. A monolitikus, magukban működő programok kora lejárt, a jövő az elosztott alkalmazásoké, melyek különböző platformokon futnak, egymással mégis szorosan kapcsolatban állnak, adatokat cserélnek. Mivel egy ilyen alkalmazáson nem egy programozói csoport dolgozik, nem is egy cég, hanem több cég által írt komponensekből tevődik össze, ezek együttműködését biztosítani kell. Ez szabványokkal, szabványos programozói interfészekkel érhetők el. Más technológiáktól független alkalmazások készítése ma már nem lehet cél.

Felhasznált irodalom

Professor Dr. Gregor von Bochmann (1983): *Distributed systems design*. Springler-Verlag Berlin Heidelberg, New York.
Csizmazia Balázs (1998): *Hálózati alkalmazások készítése*. Budapest, Kalibán BT.
Nyékyné G. Judit [szerk.] (1999): Java útikalauz programozóknak. Budapest, ELTE TTK Hallgatói Alapítvány.
Dr. Subrahmanyam Allamaraju (1999): *Nuts and Bolts of Transaction Processing*. http://www.subrahmanyam.com/articles/transactions/NutsAndBoltsOfTP.html
IBM (2002): *Websphere MQ for Windows NT and Windows 2000 Quick Beginnings Version 5.3*. (amqtac03.pdf)
IBM (2002): Websphere *MQ Application Programming Guide* (amqtac03.pdf)
Vargha Márton (1998): *Üzenetváltás programok között*. Telecomputer 3. évf. /21. sz. http://www.net.hu/telecomputer/3_21/8_1k.htm
Dr. Sugár Péter (1998) *Kereskedelmi üzenetkezelés*. http://www-5.ibm.com/hu/news/kek_rozsa/1998/02/mqseries.html
Dr. Sugár Péter (1999) *E-business és elektronikus kereskedelem*. http://www.ibm.com/hu/news/kek_rozsa/1999/01/e-business_2.html
Dr, Sugár Péter *Informatikai alkalmazások együttműködése*.
Dr, Sugár Péter *Tranzakció kezelés és az MQSeries család*. 2002. április 5. IBM Szoftver Partnertalálkozó
IBM Szoftver Partner Klub. 2002. február 21.
Kovács Mónika (1999.) *Middleware, a középréteg*. Diplomamunka.
Simon Chapman: *Message Oriented Middleware and MQSeries*. http://www.xinetica.com/tech_explained/technical/mq_series/wp_mq_series.html
The Open Group Distributed Transaction Precessing: The XA+ specification
Sharif Uddin (1999): *Survey on Middleware*. http://trident.mcs.kent.edu/~javed/DL/surveys/IAD99s-middlewares/
Java Message Server Tutorial (http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/jms_tutorialTOC.html)

Internet címek

The Source For Java Technology
http://java.sun.com
Sun Microsystems
http://www.sun.com
Sun Microsystems Magyarország
http://java.sun.hu
IBM Magyarország
http://www.ibm.hu
IBM
http://www.ibm.com
The Open Group
http://www.opengroup.org
ISO Online
http://www.iso.org