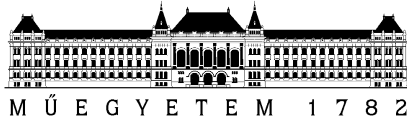


Rendszerintegráció és -felügyelet laboratórium (VIMIM309)

Kommunikáció JMS és JMX technológia segítségével

Mérési segédlet

Készítette: Hegedüs Ábel

Utolsó módosítás: 2011. március 18.

Verzió: 1.0

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

1 Bevezető

A labor során a méréseket végző hallgató a gyakorlatban is megismerkedik a rendszerintegráció és rendszerfelügyelet során használatos módszerekkel és eszközökkel. Végigköveti egy elosztott alkalmazás megvalósításának és felügyeletének legfontosabb lépéseit, ipari környezetben használt integrációs köztes réteg (middleware) technológiák és felügyeleti eszközök használatával. A mérések a következő témakörökhöz kapcsolódnak:

1. Munkafolyamatok megvalósítása Java nyelven
2. Megbízható üzenetküldés IBM WebSphere MQ alapon
3. Kommunikáció JMS és JMX technológia segítségével
4. OSGi szolgáltatások fejlesztése
5. Modell alapú eszközintegráció elosztott környezetben (SDE)
6. Szabályalapú üzleti logika
7. Üzleti folyamatok felügyelete

A jelen mérés során a hallgatók megismerkednek az üzenetsorok kezelésének szabványos módjával JMS segítségével. Ehhez kapcsolódóan betekintést nyernek a JMX keretrendszerbe, amely segítségével konfigurálhatóak az üzenetsorok és a JNDI címfeloldó technológiával, amellyel egyedi azonosítók alapján lehet referenciát kérni a megfelelő objektumokra (pl. üzenetsorok).

2 Java Message Service bevezető

Az előző mérésen használt Websphere MQ alapú üzenetsorok hátránya, hogy az alkalmazások kizárólag az IBM adott keretrendszerén keresztül tudnak kommunikálni. Ehelyett jó lenne, ha az elkészített alkalmazások tetszőleges üzenetsor kezelő keretrendszer felett futtathatóak lennének. Erre ad lehetőséget a **Java Message Service (JMS)** szabványos alkalmazásprogramozási felület (API).

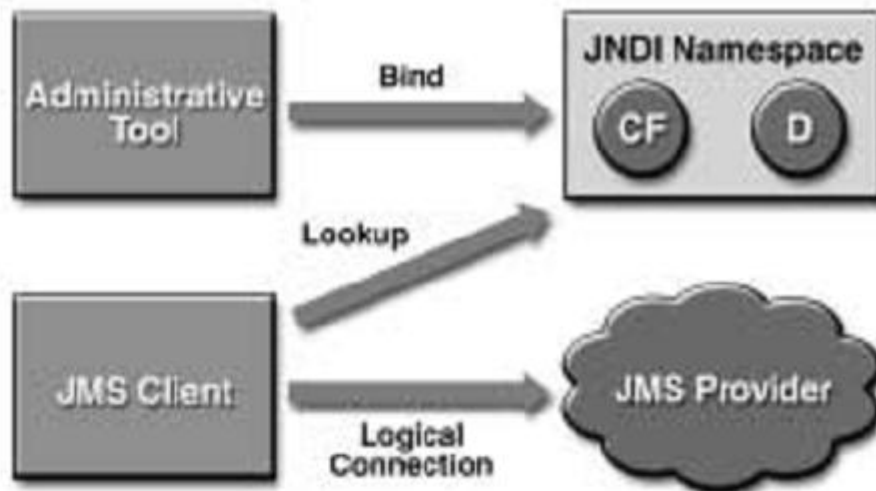
A JMS segítségével az alkalmazások képesek üzentek létrehozására, küldésére, fogadására és feldolgozására. A JMS igyekszik maximalizálni az alkalmazások hordozhatóságát különböző JMS szolgáltatók között egy adott üzenetküldő megoldásban. Segítségével megvalósítható az aszinkron és megbízható kommunikáció.

2.1 Mikor érdemes JMS API-t használni?

Egy vállalati alkalmazás szolgáltató a következő esetekben szokta az üzenetküldő APIt választani egy szorosan csatolt API helyett, ha:

- A szolgáltató azt szeretné, ha a komponensek **nem függnek más komponensek interfészeinek információtól, hogy könnyen kicserélhetők legyenek**.
- A szolgáltató szeretné. hogy az **alkalmazás futni tudjon** akkor is, **ha nem minden komponense elérhető**.
- Az alkalmazás üzleti modellje lehetővé teszi, hogy **egy komponens információkat küldjön egy másiknak és folytassa működését** anélkül, hogy azonnali választ kapna.

2.2 JMS API architektúra



1. ábra JMS API architektúra

Egy JMS alkalmazás következő részekből áll, ahogy az ábra is mutatja:

- A **JMS szolgáltató** (*provider*) implementálja a JMS interfészt, továbbá adminisztratív és vezérlési funkciókat tartalmaz.
- A **JMS kliensek** olyan Java nyelven írt programok vagy komponensek, amelyek üzeneteket termelnek és dolgoznak fel.
- Az **üzenet** olyan objektumok, amelyek a kliensek között információt visznek át.
- Az **adminisztrált objektumok** olyan előre konfigurált JMS objektumok, amelyeket egy adminisztrátor készített és a kliensek használják. Ilyenek a célállomás (*destination*) és a kapcsolatkezelők (*connection factory*).
- A **natív kliensek** olyan programok, amelyek az üzenetküldő rendszer natív API-ját használják a JMS API helyett.

2.3 Üzenetküldő megoldások

A leggyakrabban használt megoldások a pont-pont összeköttetés és a hirdető/feliratkozó módszer.

- **Pont-pont összeköttetés** (*Point-to-point, PT*P) esetén az alkalmazások az **üzenetsorok, küldők és fogadók** koncepcióira épülnek. Minden üzenetnek egy fogyasztója van. A küldőnek és a fogadónak nincsenek időzítési függései. A fogadó attól függetlenül megkaphatja az üzenetet, hogy a küldés időpontjában futott-e. A fogadó visszajelez az üzenet sikeres fogadása esetén.
- A **hirdetés/feliratkozás** (*publish/subscribe*) esetén a kliensek egy **témára** küldik az üzeneteket. A rendszer biztosítja, hogy az üzenetek eljutnak az **összes feliratkozott klienshez**. Ezáltal minden üzenetnek több fogyasztója lehet. A küldők és a feliratkozók között időzítési függőség van. A kliens, amely feliratkozik egy témára, csak olyan üzeneteket kap meg, amelyek az után érkeztek, hogy a kliens feliratkozott. Végül a kliensnek aktívnak kell maradnia ahhoz, hogy üzeneteket fogadjon.

2.4 Üzenet feldolgozás

Lehetőség van az üzeneteket szinkron és aszinkron módon fogadni. Szinkron esetben a fogadó explicit módon lekéri az üzenetet egy „*receive*” metódus segítségével. Aszinkron esetben a kliens egy „*message listener*”-t regisztrál, amely akkor fut le, ha érkezik egy üzenet. Ilyenkor a JMS szolgáltató meghívja az „*onMessage*” metódust a regisztrált üzenetkezelőben.

3 Java Naming and Directory Interface

A **Java Naming and Directory Interface (JNDI)** API egy könyvtár és elnevezési funkcionalitást szolgáltat Java alkalmazásokhoz. Segítségével az objektumokat összetett **névterekbe** sorolhatjuk, így elérhetők lesznek anélkül, hogy fordításidőben pontos referenciával rendelkeznénk hozzájuk.

A **könyvtárszolgáltatás** elérhetővé tesz különböző információkat egy hálózati környezet felhasználóiról és erőforrásairól. Ennek az információnak a reprezentálásához egy nevesítő rendszert használ a könyvtár objektumok azonosításával és rendszerezésével. Minden könyvtár objektum egy kapcsolatot jelent attribútumok és értékek között. Ezáltal a könyvtárszolgáltatás hierarchikusan és rendezetten ad leképezést érthető nevek és objektumok között.

3.1 Elnevezési alapok

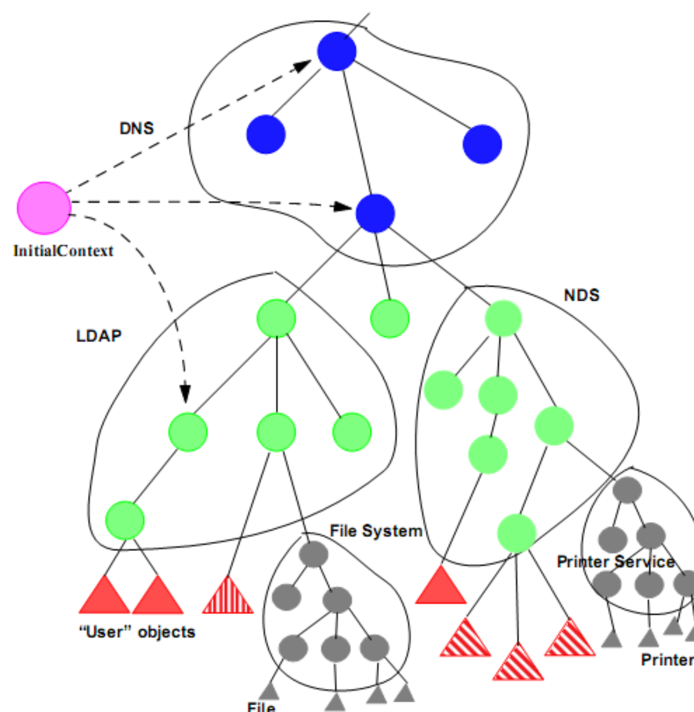
Minden **név** szintaktikus szabályok egy halmaza (*naming conventions*) alapján generálódik. Az **atomi név** egy szétválaszthatatlan része a névnek, az elnevezési szabályok szerint. Az **összetett név** egy nulla vagy több atomi név sorozatából áll elő a szabályok szerint.

Egy **asszociációt** egy atomi név és egy objektum között kötésnek (*binding*) hívunk. A **kontextus** (*context*) olyan objektum, amelynek állapota a megkülönböztethető atomi nevekkel rendelkező kötések halmaza.

Az összetett nevek feloldása a tartalmazott atomi nevek sorozatos feloldása az egymást követő kontextusokban.

Az **elnevezési rendszer** (*naming system*) az azonos típusú (azonos elnevezési konvenció alapján előálló) kontextusok összekötött halmaza, melyek mind azonos műveleteket biztosítanak. A névtér (*namespace*) az elnevezési rendszerben található összes név halmaza.

A **kompozit név** olyan név, amely több elnevezési rendszerben is szerepel. Egy rendezett listában tartalmazza az egyes rendszerekben érvényes neveket.



2. ábra Kompozit névtér példa

3.2 Könyvtár objektumok

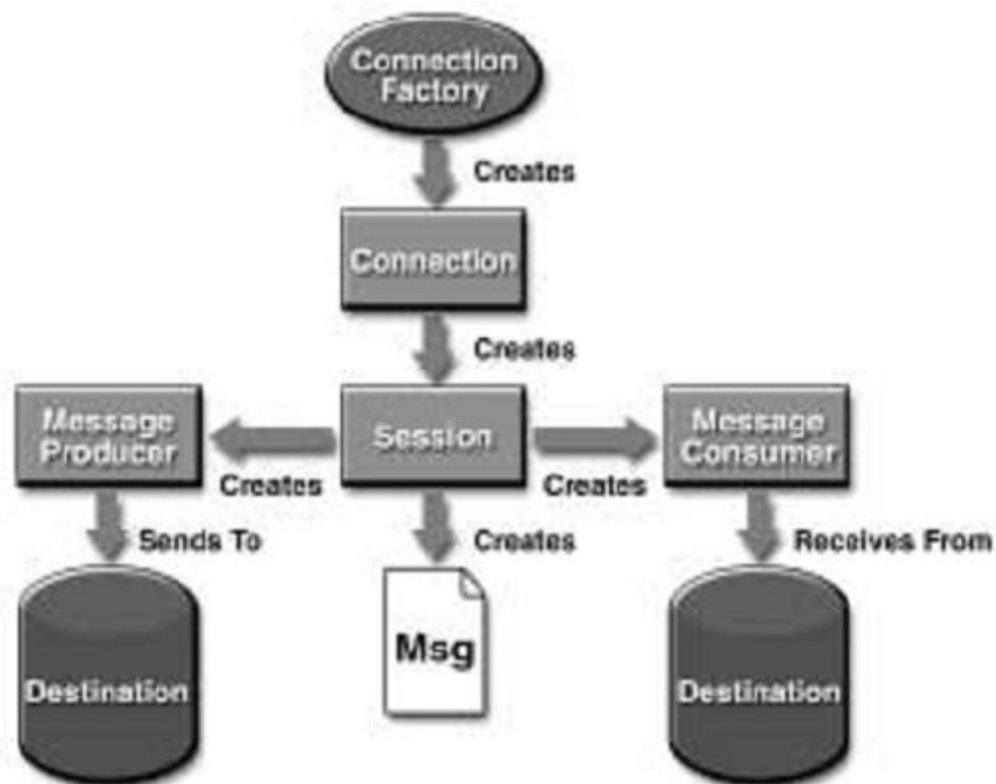
Az elnevezési rendszer elsődleges feladata **a nevek leképezése objektumokra**. Egy könyvtár objektum (*directory object*) egy bizonyos típusú objektum, amely különböző információt reprezentál egy számítási környezetben. A könyvtár objektumokhoz tartoznak attribútumok (*attributes*), amelyek egy azonosítót és értékek egy halmazát tartalmazzák.

3.3 URL-ek és kompozit nevek

Uniform Resource Locators (URLs) olyan kompozit nevek, amelyek szintakszisát az URL definíciója határozza meg. JNDI kliensek URL-eket használhatnak tetszőleges típusú objektumok elérésére.

4 JMS API programozási modell

A JMS alkalmazások alapvető építőelemei a következők: **adminisztrált objektumok, kapcsolatok, munkamenetek** (*session*), **üzenettermelők** (*producers*), **üzenetfogyasztók** (*consumers*), **üzenetek**. Az elemek közötti kapcsolatokat ábrázolja az alábbi ábra:



3. ábra JMS API progranozási modell

4.1 Kapcsolatkezelő

A kapcsolatkezelő (*connection factory*) olyan adminisztrált objektum, amelynek segítségével a kliensek kapcsolatot teremtenek a szolgáltatóval. A kapcsolatkezelő tartalmazza az adminisztrátor által definiált kapcsolat konfigurációs paraméterek halmazát.

```
Context ctx = new InitialContext();
QueueConnectionFactory queueConnectionFactory =
  (QueueConnectionFactory) ctx.lookup("QueueConnectionFactory");
```

4.2 Célállomás

A célállomás (*destination*) az az objektum, amelyet a kliens specifikál, mint a cél a küldött üzenetekhez és a forrás a feldolgozott üzenetekhez. *PTP* esetén ezeket üzenetsoroknak hívjuk.

```
Queue myQueue = (Queue) ctx.lookup("MyQueue");
```

4.3 Kapcsolatok

A kapcsolat (*connection*) valósítja meg a virtuális összeköttetést a JMS szolgáltatóval. Jelképezhet például egy nyitott *TCP/IP socket*-et a kliens és a szolgáltató *deamon* között. Egy kapcsolat több munkamenetben is használható.

```
QueueConnection queueConnection =
  queueConnectionFactory.createQueueConnection();
queueConnection.close();
```

4.4 Munkamenet

A munkamenet (*session*) egy egyszálú környezet az üzenetek létrehozására és feldolgozására. Arra használható, hogy üzenet előkészítőket, üzenet feldolgozókat és üzeneteket készítsen.

```
QueueSession queueSession =
  queueConnection.createQueueSession(true, 0);
```

Az első argumentum megadja, hogy tranzakciót használ-e a munkamenet, a második jelzi, hogy az üzenetek visszaigazolása nincs meghatározva.

4.5 Üzenet előkészítők

Az üzenet előkészítő (*message producer*) olyan munkamenet által létrehozott objektum, amellyel üzenetek küldhetőek egy adott célállomásra.

```
QueueSender queueSender = queueSession.createSender(myQueue);
queueSender.send(message);
```

4.6 Üzenet feldolgozók

Az üzenet feldolgozó (*message consumer*) olyan munkament által létrehozott objektum, amely képes egy adott célállomásra küldött üzenetek fogadására.

```
QueueReceiver queueReceiver = queueSession.createReceiver(myQueue);
queueConnection.start();
Message m = queueReceiver.receive();
```

4.7 Üzenetkezelők

Az üzenetkezelő (*message listener*) olyan objektum, amely aszinkron eseménykezelőként viselkedik üzenetekhez. A *MessageListener* interfészt valósítja meg, amelynek egyetlen metódusa van, az *onMessage*. Ebben a metódusban adható meg mi történjen egy üzenet érkezésekor.

```
QueueListener queueListener = new QueueListener();
QueueReceiver.setMessageListener(queueListener);
```

A kapcsolat start metódusát az üzenetkezelő beregisztrálása után kell elindítani, hogy ne vesszen el egy üzenet sem.

4.8 Üzenetek

A JMS üzenetek három részből állnak:

- Fejléc
- Tulajdonságok (opcionális)
- Törzs (opcionális)

4.8.1 Fejléc

Az üzenet fejléce előre meghatározott adatokat tartalmaz, amelyeket a hozzájuk tartozó *set/get* metódusokkal lehet beállítani és lekérdezni. Ezek az adatok a következők:

JMSDestination
JMSDeliveryMode
JMSExpiration
JMSPriority
JMSMessageID
JMSTimestamp
JMSCorrelationID
JMSReplyTo
JMSType
JMSRedelivered

4.8.2 Tulajdonságok

Az üzenetben létrehozhatók és beállíthatók olyan tulajdonságok, amelyekbe a **fejléc adatain kívüli további információ**t kíván a kliens elhelyezni. Így valósítható meg például kompatibilitás más üzenetalapú rendszerekkel.

4.8.3 Üzenet törzs

A JMS API öt különböző üzenet törzs típust különböztet meg, ezek:

TextMessage	java.lang.String objektum
MapMessage	név-érték párok halmaza, a nevek String objektumok, az értékek primitív típusúak
BytesMessage	bájtok folyama (stream)
StreamMessage	primitív típusú értékek folyama
ObjectMessage	Egy sorosítható objektum
Message	Nincs törzs

Üzenetek létrehozása:

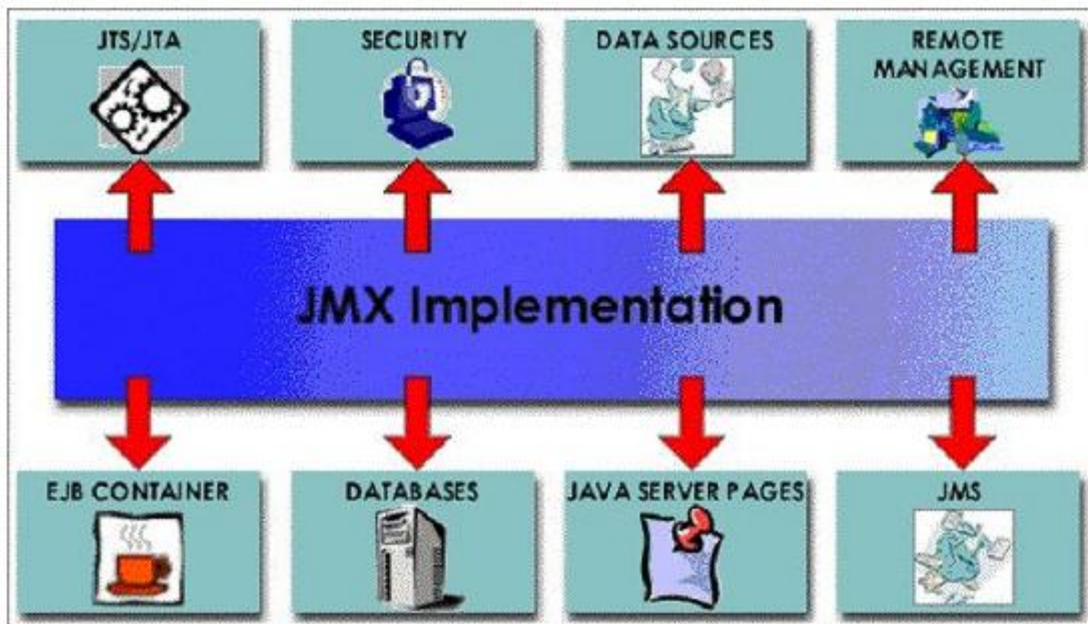
```
TextMessage message = queueSession.createTextMessage();
message.setText(msg_text);     // msg_text is a String
queueSender.send(message);
```

Üzenetek feldolgozása:

```
Message m = queueReceiver.receive();
if (m instanceof TextMessage) {
    TextMessage message = (TextMessage) m;
    System.out.println("Reading message: " + message.getText());
} else {
    // Handle error
}
```

5 JMX bevezető

A nyílt forráskódú J2EE készlet sikere nagyban köszönhető a **Java Management Extension (JMX)** technológiának. A JMX szoftverintegrációs eszköz segítségével a felhasználók egy közös gerincre építhetik a különböző modulokat, konténereket és plug-ineket. Az általános felépítés az alábbi ábrán látható:



4. ábra JMX integrációs busz és a JBosss szabványos komponensek

A JMX három fő szintet határoz meg, ezek:

- **Műszerezés** (*Instrumentation*), amely a kezelendő erőforrásokat tartalmazza. A műszerezési szint definiálja a követelményeket a JMX által kezelhető erőforrások implementációjához. Egy JMX által kezelhető erőforrás szinte bármi lehet, alkalmazás, szolgáltatás komponens, eszköz, és még sok minden. A felhasználó egy vagy több **managed bean** (vagy *MBean*) használatával valósítja meg egy adott erőforrás műszerezését. Ez a szint specifikál továbbá egy **értesítési** (*notification*) **mechanizmust** is. A műszerezési szint MBean, értesítési modell elem és MBean metadata osztály típusú objektumokat tartalmaz.
- **Ágensek** (*Agents*), amelyek a műszerezési szinten elhelyezkedő objektumok vezérlői. Az ágens szint MBean szerver és ágensszolgáltatás komponenseket tartalmaz.
- **Elosztott szolgáltatások** (*Distributed services*), amely mechanizmuson keresztül az adminisztrációs alkalmazások kapcsolatba lépnek az ágensekkel és azok kezelt objektumaikkal.

5.1 Managed Beans (MBeans)

Az MBean olyan Java objektum, amely implementálja a szabványos MBean interfészeket és követi a kapcsolódó tervezési mintákat. Az MBean interfész a következőket tartalmazza:

- **Attribútum értékek**, amelyeket név szerint le lehet kérdezni
- **Műveltek és metódusok**, amelyeket meg lehet hívni
- **Értesítések és események**, amelyeket jelezni lehet
- Az **MBean** Java osztályának **konstruktorai**

Négy MBean típust definiál a JMX, ezek:

- **Standard MBean**: sima JavaBean stílusú elnevezési konvenciót használ, statikusan definiált a management interfészen.
- **Dynamic MBean**: implementálják a DynamicBean interfészt és futásidőben ajánlják ki az interfészüket, amikor példányosítják a komponenst. Ezzel nagyobb rugalmasságot biztosítva.
- **Open MBean**: a Dynamic MBean-ek bővített változata, egyszerű, önleíró, felhasználó-barát adat típusokat használnak az egyetemes kezelhetőség érdekében.
- **Model MBean**: szintén a Dynamic MBean kiterjesztése, a ModelBean interfészt valósítja meg. Használatukkal egyszerűsödik az erőforrások műszerezése az alapértelmezett működés biztosításával.

5.2 Ágensszolgáltatások

A JMX a következő típusú szolgáltatásokat definiálja az ágensekhez:

- **Dinamikus osztálybetöltő** management applet szolgáltatás, amely lehetővé teszi az új osztályok lekérdezését és példányosítását tetszőleges hálózati helyről.
- **Monitorszolgáltatás**, amellyel az MBean-ek attribútumainak értékei figyelhetők és bizonyos változások esetén jelezhetnek más objektumoknak.
- **Időzítő szolgáltatás**, amely egyszeri riasztás vagy ismételt értesítés segítségével ad időzítési mechanizmust.
- **Kapcsolatszolgáltatás**, amely asszociációkat definiál MBean-ek között és biztosítja a konzisztenciát a kapcsolatok mentén.

6 A mérés elvégzése

A mérés során a hallgatóknak át kell alakítaniuk az második mérésen készült programjukat olyan formába, hogy a folyamat egyes lépései közötti adatáramlás üzenetsorok segítségével legyen megoldva. Az összes csomópont külön Java folyamat legyen (nem csak thread), amelyek kizárólag a JMS API-n kommunikálnak, nincs központi szinkronizációs komponens. A csomópontok között tényleges adatáramlás legyen, ne csak primitív token átadás.

A következő feladatokat kell elvégezni:

- JMX-en keresztül a folyamat csomópontjai közötti kommunikációhoz szükséges egyes üzenetsorok létrehozása.
- Java osztályok átalakítása, lehetőleg a kommunikáció elkülönítése (ahogy előző mérésen is). Általában egy get és egy put metódus megvalósítása szükséges, amely magába foglalja az JMS-el való kommunikációt.
- Folyamat lépéseinek összekötése a megfelelő ki és bemeneti sorok megadásával.

- Egyszerű grafikus megjelenítés, ahol látható az adott csomóponton elérhető adat és léptethető a folyamat, valamint elágazás esetén döntés kiválasztható.

A folyamat többször is futtatható legyen az JMX kezelő program használata nélkül (ne maradjon futást zavaró üzenet a sorokban).

6.1 A mérés kiértékelése

A mérés után leadott jegyzőkönyvben szerepeljen a mérés elvégzéséhez szükséges lépések leírása megismételhető módon. Legyen benne a megvalósított folyamat rövid leírása, azok a műveletek, amelyeket az JMX felületén el kellet végezni, a létrehozott üzenetsorok és paramétereik. Szerepeljen benne az, hogy hogyan és mennyire kellett az eredeti programot megváltoztatni, hogyan valósították meg az egyes csomópont típusokat. Tartalmazzon megfelelő mennyiségű képernyőképet és útmutatót ahhoz, hogy hogyan kell lefuttatni az elkészült programot.

7 Ellenőrző kérdések

A beugró kérdések kizárólag ebben a dokumentumban leírtakat fogják visszakérdezni, a négy linkelt leírás a mérés elvégzéséhez szükséges pluszinformációkat tartalmazzák. Figyelem, az alábbi kérdések csak példák, a beugróban más kérdések is szerepelhetnek.

1. Milyen részekből áll egy JMS alkalmazás?
2. Milyen típusú MBean-eket definiál a JMX?
3. Soroljon fel legalább négy adatot, amely a JMS üzenet fejlécében megadható?
4. Mi az elnevezési rendszer?
5. Minek a rövidítése és mire használható a JNDI?

8 Segédanyagok

Az alábbi segédanyagok egyrészt bővebb elméleti ismereteket tartalmaznak, másrészt hasznosak lehetnek a mérés elvégzése során.

The JBoss 4 Application Server Guide 2.1 fejezet: An introduction to JMX

Java™ Message Service API Tutorial 1, 2, 3 fejezetek: Overview, Basic Programming Concepts, The JMS API Programming Model

Java Naming and Directory Interface™ Application Programming Interface 3, 4, 5 fejezetek: Overview of the Architecture, Fundamentals, Overview of the Interface

Java Message Service API Architecture, JMS Message Model, JMS Common Facilities, JMS Point-to-Point Model, JMS Example Code