ISSN : 1693 - 1173

INTEGRASI JAVA MESSAGE SERVICE (JMS) DENGAN MESSAGE-DRIVEN BEAN PADA TEKNOLOGI JAVA EE

Wiharto [6)]

Abstract

There are three a part of Enterprise Java Bean (EJB) architecture in the Java Enterprise Edition (Java EE) technology. First, session bean, entity bean and the last message driven bean (MDB). Session bean is a bean that can be accesed by remote method invocation (RMI) for give ability, it provided business interface. Entity bean representing and manipulating data which persistent in that application. For acces that entity bean, it also provided business interface as the same with session bean. Message driven bean is one of component in the EJB which has an unique characteristic, it is whitout bisnis interface, so client can not be accesed with the RMI, it can be only accesed with the messaging system like java message service (JMS).

***Kay words** : EJB, RMI, Java Message Service.*

I. Pendahuluan

Arsitektur aplikasi enterprise modern, layer of service dibagi menjadi 3 layer yaitu presentation, bussines dan data provider. Java EE dalam mengimplementasikan layer presentasi menggunakan teknologi Servlet, JSP, JSF, JSTL atau Swing. Untuk layer bussines menggunakan teknologi Enterprise Java Bean , EJB merupakan teknologi dalam Java EE untuk penyediaan bussines layer. Untuk arsitektur EJB yang khusus digunakan dalam bussines layer, terbagi lagi menjadi 2 sub-layer yaitu proses bisnis dan persisten. Layer persistent merupakan layer yang bertugas melakukan mapping dari Database Relational ke Obejek (Untuk database Relational). Proses bisnis akan diimplementasikan dengan menggunakan Session Beans dan Message-Driven Beans, sedangkan untuk persitens data menggunakan Entities bean.

Pada Enterprise Java Bean komponen Session Bean dan Entity Bean dapat diakses oleh client dengan menggunakan RMI, karena

[6)] Staf Pengajar Universitas Sebelas Maret Surakarta

kedua komponen tersebut mempunyai bisnis interface yang dapat diakses secara local maupun remote. Untuk komponen Message-Driven Beans (MDB), client tidak dapat mengakses dengan menggunakan RMI karena komponen ini tidak mempunyai bisnis interface (baik local maupun remote). Satu-satunya cara untuk mengakses komponen tersebut adalah dengan menggunakan konsep messaging. Messaging menggunakan konsep, bahwa antara client dan server terdapat layer lagi yaitu Middleman, Middleman ini berfungsi menerima pesan dari satu atau labih message producer dan mem-broadcast-nya ke satu atau lebih message consumer. Salah satu teknologi yang menggunakan konsep messaging adalah JMS.

II. Rumusan Masalah

Bagaimana mengakses atau mengintegrasikan komponen Message Driven Bean pada Java EE dengan menggunakan messaging JMS.

III. Tujuan

Penelitian ini bertujuan untuk mengetahui bagaiman teknologi messaging JMS mengakses komponen Message Driven Bean.

IV Metode Penelitian

Metode yang dipakai dalam penelitian ini dengan menggunakan metode pustaka. Kajian dimulai dengan kajian tenteng teknologi messaging JMS dengan berbagai mode messaging-nya yang disertai dengan sample programnya. Tahapan selanjutnya kajian terhadap teknologi Java EE dengan komponen Message Driven Bean dan integrasi komponen tersebut dengan JMS.

V. Pembahasan

a. Konsep Messaging dalan Java Message Service

Messaging merupakan alternative RMI, gagasan dibalik messaging adalah *middleman* site yang terletak antara client dan server. Middleman menerima pesan dari satu atau labih message producer dan mem-broadcast-nya ke satu atau lebih message consumer. Oleh karena itu produser dapat mengirimkan pesan kemudian melanjutkan proses berikutnya. Dia juga dapat dipilih sebagai suatu notified ketika consumer telah selesai meresponnya. Hal seperti ini disebut sebagai suatu pemrograman asinkronouse.

Protokol yang digunakan dalam messaging adalah RMI-IIOP (*Internet Inter-ORB Protocol*). Salah satu teknologi yang menggunakan konsep messaging adalah JMS. JMS mempunyai dua bagian yaitu API untuk membuat/menulis coding mengirim dan menerima message dan *Service Provider Interface* (SPI) yang sudah plug-in dalam JMS Provider. Dalam JMS API terdiri dari interface dan class yang mensuport dua mode dasar messaging yaitu [5][2][7]:

1. ***Point-to-point***. Producer message mengirimkan sebuah message ke ***queue***. Message dibaca sekali oleh consumer. Setelah message tersebut dibaca kemudian akan di hapus dari ***queue***.
2. ***Publish/subscribe***. Produser message mengirim sebuah message ke ***topic***. Consumer satu demi satu melakukan register dengan atau ke subscribe ke sebuah ***topic***. Message yang tinggal pada ***topic*** sampai semua consumer teleh membaca messagenya. Jika seorang consumer tidak ter-register pada ***topic*** pada saat message terkirim maka hal ini akan menyebabakn message tidak dapat dibaca secara *subsequent*.

Suatu queue atau topic dapat menjadi syncronous atau asyncronous. Jika suatu consumer syncronous dan tidak ada message pada topic atau queue, maka consumer akan menggantung sampai message terkirim. Consumer dikatakan asyncronous jika telah terdaftar sebagai pendengar/listener peda topic atau queue, maka ketika message terkirim pendengar/listener akan ternotified dan consumer asyncronous akan membaca message.

JMS tidak menjamin suatu message terkirim sesuai perintah. Maka ketika produser mengirim sebagian message A diikuti sebagian message B, kita tidak dapat mengasumsikan bahwa consumer akan membaca bagian A sebelum bagian B. Ada beberapa langkah queue atau topic yang harus producer jalankan dan sejumlah step/langkah tersebut bergantung pada apakah consumer itu Synchronous atau Asynchronous. Untuk lebih jelasnya nanti bisa kita lihat dalam contoh. Sebelum kita ke contoh aplikasi yang menggunakan JMS, kita tunjukkan terlebih dahulu langkah-langkah dalam pemrograman JMS berikut [5] :

1. Lokasi Provider Java Message Service instance ConnectionFactory. Pertama kita akan membutuhkan akses ke dalam provider JMS dari bagian produk MOM yang kita

gunakan. Untuk itu, kita membutuhkan koneksi yang baik dengan menggunakan instance *ConnectionFactory* dengan cara mencarinya dalam JNDI. Suatu administrator secara selektif akan menciptakan dan mengkonfigurasikan *ConnectionFactory* untuk digunakan oleh client JMS.

2. Menciptakan suatu Koneksi JMS. Suatu koneksi JMS merupakan koneksi yang aktif pada provider JMS, yang mengelola jaringan komunikasi pada level rendah, seperti pada koneksi JDBC. Kita menggunakan *ConnectionFactory* untuk memperoleh koneksi.
3. Menciptakan suatu session JMS. Session JMS merupakan suatu objek pembantu yang digunakan ketika mengirim dan menerima pesan. Ini bertindak sebagai factory/pabrik untuk consumer dan producer message, dan juga memudahkan kita untuk meng-encapsulasi message kita dalam transaksi.
4. Meletakkan tujuan JMS. Tujuan JMS merupakan chanel yang dituju untuk mengirim atau dari mana message diterima. Lokasi tujuan yang benar dapat dianalogikan dengan suatu chanel yang tepat ketika menonton televisi atau menjawab telepon, sehinggga kita akan mendapatkan message yang tepat kita inginkan.
5. Membuat producer atau consumer JMS. Jika kita ingin mengirimkan suatu pesan, maka kita harus memanggil object JMS untuk melakukan pengiriman pesan itu. Object ini disebut producer. Untuk menerima pesan harus memanggil object JMS dan memerintahkannya. Object ini disebut object consumer.
6. Menerima atau mengirimkan pesan. Jika berada pada posisi producer, maka langkah pertama harus meletakkan pesan secara bersamaan. Ada banyak perbedaan type message seperti text, bytes, streams, object dan maps. Setelah meng-instance message, selanjutnya dapat mengirimkan message itu menggunakan object producer.

Contoh : JMS Syncronouse dan Asyncronouse

Contoh berikut untuk menjelaskan cara kerja JMS dengan maode messaging ***point-to-point***, yaitu untuk mengirimkan message ke *queue* dan mengambil message dari *queue* oleh customer. Berikut potongan programnya :

1. Mengirim message ke dalam Queue, berikut potongan listing programnya :

```
public class MessageSender {
 @Resource(mappedName = "jms/MaswieBukuConnectionFactory")
 private static ConnectionFactory connectionFactory;
 @Resource(mappedName = "jms/MaswieBukuQueue")
 private static Queue queue;
 public void produceMessages() {
 MessageProducer messageProducer;
 TextMessage textMessage;
 try  {
    Connection connection = connectionFactory.createConnection();
  Session session = connection.createSession(false,
  Session.AUTO_ACKNOWLEDGE);
  messageProducer = session.createProducer(queue);
  textMessage = session.createTextMessage();
   textMessage.setText("Test, dapat Mendengar Infromasi ini.....?");
  System.out.println("Send Informasi : " + textMessage.getText());
  messageProducer.send(textMessage);
  messageProducer.close();
  session.close();
  connection.close();
 }  catch (JMSException e) {    e.printStackTrace();    }
 }
 public static void main(String[] args) {
     new MessageSender().produceMessages();
 }}
```

2. Mengambil message dari Queue, berikut listing programnya :

```
public class MessageReceiver
{
 @Resource(mappedName = "jms/MaswieBukuConnectionFactory")
 private static ConnectionFactory connectionFactory;
 @Resource(mappedName = "jms/MaswieBukuQueue")
 private static Queue queue;
 public void getMessages()  {
  Connection connection;
  MessageConsumer messageConsumer;
  TextMessage textMessage;
  boolean sukses = false;
  try  {
     connection = connectionFactory.createConnection();
```

```
            Session session = connection.createSession(false,
            Session.AUTO_ACKNOWLEDGE);
            messageConsumer = session.createConsumer(queue);
            connection.start();
            while (!sukses)
            {
               System.out.println("Tunggu Message diambil dari queue....");
               textMessage = (TextMessage) messageConsumer.receive();
   if (textMessage != null)
   {
      System.out.print("Message diterima/diambil : ");
      System.out.println(textMessage.getText());
    }
    if          (textMessage.getText()          !=          null                    &&
textMessage.getText().equals("OkeMaswie!"))
    {
        sukses = true;
    }
 }
  messageConsumer.close();
  session.close();
  connection.close();
  }catch (JMSException e) {  e.printStackTrace();   }
}
 public static void main(String[] args)       {
          new MessageReceiver().getMessages();
  }}
```

Berdasarkan contoh tersebut ada beberapa statemen program yang perlu mendapat perhatian diantaranya pada saat pembuatan session yaitu :

- connection.createSession(false,Session.AUTO_ACKNOWLEDGE);

  Dalam parameter createSession ada dua, yang pertama Boolean merupakan indikasi bahwa session adalah transaction. Jika parameter tersebut kita berikan nilai true, beberapa message dapat dikirim per bagian transaction dengan memanggil method commint() dari object session dan dapat dibatalkan pula dengan memanggil method rollback(). Sedangkan parameter yang kedua mengidikasikan bagaimana cara mengirimkan *acknowledged*

(ACK) setelah message diterima. Ada tiga cara yang dapat dilakukan yaitu [6] :

a. Session.AUTO_ACKNOWLEDGE, Mengidikasikan bahwa session akan secara otomatis memanggil method acknowledge()/mengirimkan suatu ACK setetalah message diterima.
b. Session.CLIENT_ACKNOWLEDGE, Mengidikasikan bahwa session akan mengirimkan ACK tetapi dengan cara memanggil secara eksplisit method acknowledge() setetalah message diterima.
c. Session.DUPS_OK_ACKNOWLEDGE, Mengidikasikan bahwa session akan mengirimkan ACK tetapi secara lambat. Sehingga dengan asumsi tersebut message yang dikirimkan lebih dari satu.

- Dalam JMS type data yang digunakan ada lima jenis yaitu

Table 1 : Type data yang digunakan dalam JMS [6]

Type Message	Diskripsi
BytesMessage	Membolehkan pengiriman message array bytes
MapMessage	Membolehkan pengiriman message yang mengimplementasikan java.util.Map
ObjectMessage	Membolehkan pengiriman message yang berupa objek java yang mengimplementasikan java.io.Serializable
StreamMessage	Membolehkan pengiriman message array bytes, tetapi berbeda dengan BytesMessage.
TextMessage	Membolehkan pengiriman message yang berupa java.lang.String

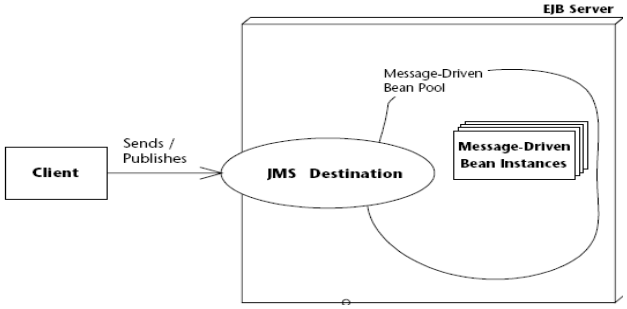
Dalam contoh tersebut untuk model pengambilan message dari queue menggunakan perintah *MessageConsumer.receive()* mempunyai kelemahan yaitu, akan melakukan blocking eksekusi yang lain sampai message diterima dari queue, atau dikenal dengan menggunakan method S*yncronouse*. Kita dapat mengatasi kelemahan tersebut dengan cara menerima message dengan model *Asyncronouse*, yaitu dengan mengimplementasikan interface javax.jms.MessageListener [4]. Interface tersebut mempunyai single method yaitu onMessage().

```
public class ContohListener implements MessageListener {
   public void onMessage(Message message) {
            TextMessage textMessage = (TextMessage)message;
            try {
                  System.out.print("Message yang diterima: ");
```

```
                System.out.println(textMessage.getText());
            } catch (JMSException e){    e.printStackTrace();    }
        }
}
```

b. Integrasi JMS dengan Message Driven Bean

Integrasi JMS dengan EJB merupakan suatu kombinasi ide. Dimana Ide tersebut adalah membolehkan Komponen EJB memanfaatkan nilai proporsional messeging seperti client *non-bloking* dan *multinary communications*. Message-driven bean merupakan komponen EJB yang special di mana dapat menerima pesan JMS sebaik tipe message yang lain. Suatu message-driven bean terbangun dari beberapa client yang mengirim message kepadanya. Client tidak dapat mengakses suatu message-driven bean melalui bisnis interface. Kenyataannya client tidak dapat mengidentifakasi message-driven bean dan secara lengsung untuk berinteraksi dengan semuanya itu. Jalan satu-satunya agar client dapat berinteraksi dengan message driven bean adalah melalui system **messaging**. Maka kita harus menggunakan provider message API yang spesifik seperti JMS, untuk mengirim message dari client yang kemudian message-driven bean akan menerima message tersebut. Dimana untuk alur prosesnya dapat diitunjukan pada gambar berikut :



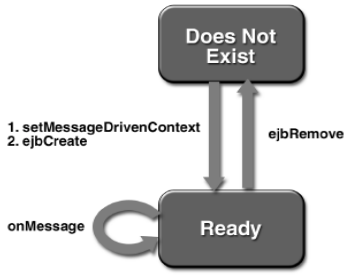
Gambar 1 : Proses pemanggilan MDB menggunakan JMS [5]

Message-driven beans mempunyai karakteristik sebagai berikut :

1. Message driven beans tidak mempunya local dan remote business interface, sehingga tidak dapat memenggil message driven beans tersebut dengan menggunakan object oriented *Remote Method Invocation* (RMI).

2. Message driven bean mensuport method generic listener terhadap message yang dikirimkan dari client.
3. Message-driven beans dieksekusi selama menerima message dari client tunggal.
4. Memproses secara Asyncronouse
5. Bersifat stateless, atau tidak memelihara assosiasi dengan suatu client
6. Message-driven beans adalah *single Threaded*. Single message-driven beans dapat memproses hanya satu message dalam suatu waktu.

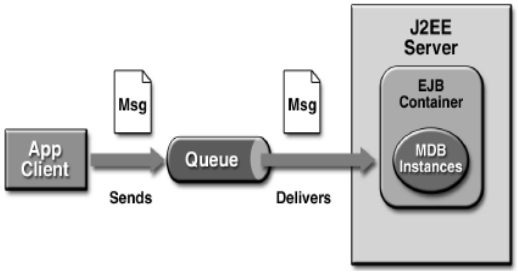
Life cycle message-driven beans ditunukkan pada gambar 2. Jika kita perhatikan dalam gambar, maka ketika sebuah message datang container akan memanggil message-driven beans yaitu method **onMessage()** untuk memproses pesan. Method **onMessage()** biasanya casting satu dari lima type JMS message dan terlepas dari kesepakatan dengan aplikasi bisnis logic. Method onMessage() dapat memanggil method pendukung, atau dapat memanggil session beans atau entity beans untuk memproses informasi di dalam message atau menyimpan dalam suatu database.



Gambar 2 : Diagram state Message-driven Beans [1]

Contoh Aplikasi Integrasi JMS dengan MDB

Untuk lebih memahami integrasi Message-driven beans (MDB) dengan JMS, perhatikan contoh berikut : sebuah aplikasi client yang menggunakan JMS producer mengirimkan message ke queue, kemudi3an message yang ada di queue akan menjadi masukkan method **onMessage(Message msg)** yang berada dalam Message-driven beans dan sekaligus mengeksekusi method tersebut. Sebagai gambaran perhatikan gambar ini



Gambar 3 : Ilustrasi integrasi JMS dan MDB [1]

Berikut tahapan dan potongan listing program aplikasi tersebut

1. Buat aplikasi JMS untuk producer-nya, yang nantinya akan berposisi sebagai client. Berikut programnya :

```
@Stateless
public class BankProducer {
@Resource(mappedName = "jms/MaswieBukuConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/MaswieBukuQueue")
private static Queue queue;
public void BankProducer(){}
public void addCustomer(String lnama)  {
try  {
Connection connection = connectionFactory.createConnection();
Session session =
 connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
MessageProducer messageProducer =
session.createProducer(queue);
TextMessage obj = session.createTextMessage();
obj.setText(lnama);  messageProducer.send(obj);
messageProducer.close();
session.close(); connection.close();
}  catch (JMSException e){   e.printStackTrace(); }
}
public static void main(String[] args) {
    new BankProducer().addCustomer("Wiharto"); }
}
```

2. Langkah selanjutnya adalah membuat EJB Message-driven beans dengan coding sebagai berikut :

```
@MessageDriven(activationConfig = {
@ActivationConfigProperty(propertyName =
"destinationType",propertyValue = "javax.jms.Queue") },
mappedName="jms/MaswieBukuQueue" )
public class ProcessCustomerBean implements MessageListener {
public ProcessCustomerBean(){}
        public void onMessage(Message arg0) {
         try {
            if (arg0 instanceof TextMessage){
           TextMessage oke = (TextMessage) arg0;
           System.out.println(oke.getText());
           System.out.println("sukses");   }
        else {    System.out.println("Type data salah");  }
        } catch(Exception e){  throw new EJBException(e);  }
}}
```

VI. Kesimpulan

Messaging merupakan alternative RMI. Gagasan dibalik messaging adalah *middleman* site yang terletak antara client dan server. Middleman menerima pesan dari satu atau labih message producer dan mem-broadcast-nya ke satu atau lebih message consumer. Dalam Enterprise Java Bean komponen MessageDriven Bean tidak mempunyai bisnis interface, sehingga untuk mengakses komponen tersebut client tidak dapat menggunakan RMI seperti pada saat mengakses komponen session bean dan entity bean. Jalan satu-satunya agar client dapat berinteraksi dengan message-driven bean adalah melalui system messaging, salah satu system message adalah JMS. Client yang akan mengakses komponen message-driven bean mengirimkan message ke queue kemudian message yang berada dalam queue tersebut menjadi masukan sebuah method yang ada dalam message-driven bean, dan sekaligus mengeksekusinya.

Daftar Pustaka

[1]. Stephanie Bodoff, Dale Green, dkk, 2002. ***"The J2EE Tutorial"***. Sun Microsystems, Inc.

[2]. Paul Giotta dkk.,2000. ***"Professional JMS Programming"***.Apress.

[3]. Raghu R. Kodali & Jonathan Wetherbee, 2006. ***"Beginning EJB 3 Application Development: From Novice to Professional"***.Apress. USA.
[4]. Michael Sikora, 2008. ***"EJB 3 Developer Guide : A Practical Guide for developers and architects to the Enterprise Java Beans Standard"***. PACKT Publishing. Birmingham. UK.
[5]. Rima Patel Sriganesh, Gerald Brose, dkk, 2006. ***"Mastering enterprise JavaBeans 3.0"***. Wiley Publishing, Inc. Indianapolis.
[6]. A David R. Heffelfinger, 2007. ***"Java EE 5 Development using GlassFish Application Server"***. PACKT Publishing. Birmingham. UK.
[7]. H.M.Deitel & dkk. 2001***."Advanced Java 2 Platform How to Program"***. Prentice Hall.