Laboratorium Projektowania Systemów Rozproszonych

Michał Kalewski

$Id: domino_lab.lyx,v 1.8 2005/11/15 07:21:46 mkalewski Exp $

30 listopada 2005 roku

Spis treści

Rozdział 1

Enterprise Java Beans

1.1 Bezstanowe komponenty sesyjne

1. Implementacja bezstanowego komponentu sesyjnego "Hello World!" – posłuż się przykładem (listingi: 1.1, 1.2, 1.3, 1.4, a także: deskryptor rozmieszczenia dla serwera JBoss – plik `ejb-jar.xml`, listing 1.5 oraz plik ustawień dla JNDI – `jndi.properties`, listing 1.6).
2. Struktura katalogów (rysunek 1.1), kompilowanie komponentów oraz klienta, ładowanie komponentów do serwera, uruchamianie klientów.
3. Odczyt ustawień JNDI w kodzie klienta (listing 1.7).
4. Interfejsy lokalne.

Rysunek 1.1: Struktura katalogów dla przykładów

```
.
|-- assemble
|
|-- etc
|
|-- src
|   |-- client
|   |   `-- pl
|   |       `-- mkalewski
|   |           `-- ejb
|   |-- server
|   |   `-- pl
|   |       `-- mkalewski
|   |           `-- ejb
|   `-- shared
|       `-- pl
|           `-- mkalewski
|               `-- ejb
`-- target
    |-- client
    |   `-- pl
    |       `-- mkalewski
    |           `-- ejb
    `-- server
        |-- META-INF
        |
        `-- pl
            `-- mkalewski
                `-- ejb
```

Listing 1.1: Interfejs zdalny – komponent sesyjny bezstanowy

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
//(Remote) Component interface
public interface Hello extends EJBObject
{
   public String SayHello() throws RemoteException;
}
```

Listing 1.2: Interfejs domowy – komponent sesyjny bezstanowy

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
//(Remote) Home interface
public interface HelloHome extends EJBHome
{
   public Hello create() throws CreateException, RemoteException;
}
```

Listing 1.3: Implementacja klasy – komponent sesyjny bezstanowy

```
package pl.mkalewski.ejb;
import javax.ejb.*;
//Implementation class for EJB
public class HelloBean implements SessionBean
{
   public void ejbCreate()
   {System.out.println("[HelloBean] ejbCreate executed");}
   public void ejbActivate()
   {System.out.println("[HelloBean] ejbActivate executed");}
   public void ejbPassivate()
   {System.out.println("[HelloBean] ejbPassivate executed");}
   public void ejbRemove()
   {System.out.println("[HelloBean] ejbRemove executed");}
   public void setSessionContext(SessionContext cs)
   {System.out.println("[HelloBean] setSessionContext executed");}
   public String SayHello()
   {
      System.out.println("[HelloBean] SayHello executed");
      return "Hello world!";
   }
}
```

Listing 1.4: Klient dla przykładowego komponentu sesyjnego

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
import javax.rmi.*;
import javax.naming.*;
public class HelloClient
{
   public static void main(String[] args) throws Exception
```

```
    {
10      Context con = new InitialContext();
        Object obj = con.lookup("Hello");
12      HelloHome hh = (HelloHome)
                    PortableRemoteObject.narrow(obj, HelloHome.class);
14      Hello h = hh.create();
        String text = h.SayHello();
16      System.out.println(text);
        h.remove();
18  }
}
```

Listing 1.5: Deskryptor rozmieszczenia – komponent sesyjny bezstanowy

```
<?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE ejb-jar PUBLIC
          "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans2.0//EN"
4         "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
6 <enterprise-beans>
  <session>
8   <ejb-name>Hello</ejb-name>
    <home>pl.mkalewski.ejb.HelloHome</home>
10  <remote>pl.mkalewski.ejb.Hello</remote>
    <ejb-class>pl.mkalewski.ejb.HelloBean</ejb-class>
12  <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
14  </session>
 </enterprise-beans>
16 </ejb-jar>
```

Listing 1.6: Ustawienia JNDI

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
2 java.naming.provider.url=jnp://localhost:1099
```

Listing 1.7: Ustawienia JNDI w kodzie aplikacji

```
Properties JNDI_PROPERTIES = new Properties();
2 JNDI_PROPERTIES.put(InitialContext.INITIAL_CONTEXT_FACTORY,
                      "org.jnp.interfaces.NamingContextFactory");
4 JNDI_PROPERTIES.put(InitialContext.PROVIDER_URL, "192.168.0.3:1099");
InitialContext con = new InitialContext(JNDI_PROPERTIES);
```

1.2 Stanowe komponenty sesyjne

1. Implementacja stanowego komponentu sesyjnego (modyfikacja pliku `ejb-jar.xml` – listing 1.8), realizującego prosty licznik.

2. Wywoływanie komponentu stanowego przez komponent bezstanowy.

3. Wywoływanie komponentu za pomocą interfejsów lokalnych (lokalny interfejs komponentu – listing 1.9, lokalny interfejs domowy komponentu – listing 1.10 oraz deskryptor rozmieszczenia – listing 1.11).

Listing 1.8: Modyfikacja deskryptora rozmieszczenia dla komponentu stanowego

```
<session-type>Stateful</session-type>
```

Listing 1.9: Interfejs lokalny komponentu

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
//(Local) Component interface
public interface CountLocal extends EJBLocalObject
{
   public int counter();
}
```

Listing 1.10: Lokalny interfejs domowy

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
//(Local) Home interface
public interface CountHomeLocal extends EJBLocalHome
{
   public CountLocal create(int init) throws CreateException;
}
```

Listing 1.11: Deskryptor rozmieszczenia – komponent bezstanowy wykorzystujący interfejsy lokalne komponentu stanowego

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC
         "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans2.0//EN"
         "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
 <enterprise-beans>
  <session>
   <ejb-name>Count</ejb-name>
   <home>pl.mkalewski.ejb.CountHome</home>
   <remote>pl.mkalewski.ejb.Count</remote>
   <local-home>pl.mkalewski.ejb.CountHomeLocal</local-home>
   <local>pl.mkalewski.ejb.CountLocal</local>
   <ejb-class>pl.mkalewski.ejb.CountBean</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
  </session>
  <session>
   <ejb-name>Hello</ejb-name>
   <home>pl.mkalewski.ejb.HelloHome</home>
   <remote>pl.mkalewski.ejb.Hello</remote>
   <ejb-class>pl.mkalewski.ejb.HelloBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <ejb-local-ref>
    <ejb-ref-name>CountHomeLocal</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
```

```
    <local-home>pl.mkalewski.ejb.CountHomeLocal</local-home>
    <local>pl.mkalewski.ejb.CountLocal</local>
    <ejb-link>Count</ejb-link>
   </ejb-local-ref>
  </session>
 </enterprise-beans>
</ejb-jar>
```

1.3 Komponenty encyjne CMP

1. Implementacja prostego komponentu do przechowywania i selekcji danych (listingi: 1.12, 1.13, 1.14, 1.15, 1.16).
2. EJB Query Language (EJB-QL).
3. Transakcje (Required, RequiresNew, Mandatory, Supports, NotSupported, Never) – listing 1.17.

Listing 1.12: Interfejs zdalny – komponent sencyjny CMP

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
//(Remote) Component interface
public interface CMPExample extends EJBObject
{
   public java.lang.Integer getId() throws RemoteException;
   public String getField() throws RemoteException;
   public void setField(String Field) throws RemoteException;
}
```

Listing 1.13: Interfejs domowy – komponent sencyjny CMP

```
package pl.mkalewski.ejb;
import javax.ejb.*;
import java.rmi.*;
import java.util.Collection;
//(Remote) Home interface
public interface CMPExampleHome extends EJBHome
{
   public CMPExample create(java.lang.Integer Id, String Field)
                                    throws CreateException, RemoteException;
   public CMPExample findByPrimaryKey(java.lang.Integer Id)
                                    throws FinderException, RemoteException;
   public Collection findByField(String Field)
                                    throws FinderException, RemoteException;
   public Collection findAll() throws FinderException, RemoteException;
}
```

Listing 1.14: Implementacja klasy – komponent encyjny CMP

```
package pl.mkalewski.ejb;
import javax.ejb.*;
//Implementation class for EJB
```

```
public abstract class CMPExampleBean implements EntityBean
{
 private EntityContext con;
 public abstract java.lang.Integer getId();
 public abstract void setId(java.lang.Integer Id);
 public abstract String getField();
 public abstract void setField(String Field);
 public java.lang.Integer ejbCreate(java.lang.Integer Id, String Field)
                                                    throws CreateException
 {
    System.out.println("[CMPExampleBean] ejbCreate executed");
    setId(Id);
    setField(Field);
    return Id;
 }
 public void ejbPostCreate(java.lang.Integer Id, String Field)
 {System.out.println("[CMPExampleBean] ejbPostCreate executed");}
 public void ejbActivate()
 {System.out.println("[CMPExampleBean] ejbActivate executed");}
 public void ejbPassivate()
 {System.out.println("[CMPExampleBean] ejbPassivate executed");}
 public void ejbRemove()
 {System.out.println("[CMPExampleBean] ejbRemove executed");}
 public void setEntityContext(EntityContext ec)
 {
  System.out.println("[CMPExampleBean] setEntityContext executed");
  this.con = ec;
 }
 public void unsetEntityContext()
 {
  System.out.println("[CMPExampleBean] unsetEntityContext executed");
  this.con = null;
 }
 public void ejbLoad()
 {System.out.println("[CMPExampleBean] ejbLoad executed");}
 public void ejbStore()
 {System.out.println("[CMPExampleBean] ejbStore executed");}
}
```

Listing 1.15: Klient dla przykładowego komponentu encyjnego

```
import javax.ejb.*;
import java.rmi.*;
import javax.rmi.*;
import javax.naming.*;
import java.util.*;
public class CMPExampleClient
{
   public static void main(String[] args) throws Exception
   {
      CMPExampleHome h = null;
      java.lang.Integer pk1 = new Integer(1);
      java.lang.Integer pk2 = new Integer(2);
      java.lang.Integer pk3 = new Integer(3);
```

```
14        try
          {
16           Context con = new InitialContext(System.getProperties());
             h = (CMPExampleHome)
18           javax.rmi.PortableRemoteObject.narrow(
                        con.lookup("CMPExample"), CMPExampleHome.class);
20
             h.create(pk1, "Wiktoria");
22           h.create(pk2, "Patrycja");
             h.create(pk3, "Michal");
24           Iterator i = h.findByField("Michal").iterator();
             System.out.println("Wyszukiwanie po slowie >>Michal<<");
26           while (i.hasNext())
             {
28              CMPExample field = (CMPExample)
                javax.rmi.PortableRemoteObject.narrow(i.next(),
30                                                    CMPExample.class);
                System.out.println(field.getId() + " " + field.getField());
32           }
             i = h.findAll().iterator();
34           while(i.hasNext())
             {
36              CMPExample field = (CMPExample)
                javax.rmi.PortableRemoteObject.narrow(i.next(),
38                                                    CMPExample.class);
                field.remove();
40           }
          }
42        catch(Exception e) { e.printStackTrace(); }
          finally {}
44     }
   }
```

Listing 1.16: Deskryptor rozmieszczenia – komponent sesyjny bezstanowy

```
   <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE ejb-jar PUBLIC
           "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans2.0//EN"
4          "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
   <ejb-jar>
6   <enterprise-beans>
    <entity>
8     <ejb-name>CMPExample</ejb-name>
      <home>pl.mkalewski.ejb.CMPExampleHome</home>
10    <remote>pl.mkalewski.ejb.CMPExample</remote>
      <ejb-class>pl.mkalewski.ejb.CMPExampleBean</ejb-class>
12    <transaction-type>Container</transaction-type>
      <persistence-type>Container</persistence-type>
14    <prim-key-class>java.lang.Integer</prim-key-class>
      <abstract-schema-name>CMPExampleTable</abstract-schema-name>
16    <cmp-version>2.x</cmp-version>
      <cmp-field><field-name>Id</field-name></cmp-field>
18    <cmp-field><field-name>Field</field-name></cmp-field>
      <reentrant>False</reentrant>
```

```
    <primkey-field>Id</primkey-field>
    <query>
     <query-method>
      <method-name>findByField</method-name>
      <method-params><method-param>
       java.lang.String
      </method-param></method-params>
     </query-method>
     <ejb-ql>
      <![CDATA[SELECT OBJECT(a)
          FROM CMPExampleTable AS a WHERE a.Field = ?1]]>
     </ejb-ql>
    </query>
    <query>
     <query-method>
      <method-name>findAll</method-name>
      <method-params></method-params>
     </query-method>
     <ejb-ql>
      <![CDATA[SELECT OBJECT(a)
          FROM CMPExampleTable AS a WHERE a.Id IS NOT NULL]]>
     </ejb-ql>
    </query>
  </entity>
 </enterprise-beans>
</ejb-jar>
```

Listing 1.17: Modyfikacja deskryptora rozmieszczenia do obsługi transakcji

```
<assembly-descriptor>
 <container-transaction>
  <method>
   <ejb-name>CMPExample</ejb-name>
   <method-name>*</method-name>
  </method>
  <trans-attribute>Required</trans-attribute>
 </container-transaction>
</assembly-descriptor>
```

1.4 Komponenty sterowane komunikatami

1. Implementacja aplikacji wymiany komunikatów z wykorzystaniem JMS API (komunikacja typu *point-to-point*, listingi: 1.18, 1.19; oraz komunikacja typu *publish/subscribe*, listngi: 1.20, 1.21).

2. Implementacja komponentu sterowanego komunikatami.

Listing 1.18: Producent wiadomości – komunikacja typu point-to-point

```
package pl.mkalewski.jms;
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;
```

```
public class Producer {
 public static void main(String[] args) {
  String messages[] = { "MESSAGE A", "MESSAGE B", "MESSAGE C",
    "MESSAGE D" };
  QueueConnectionFactory queueConnectionFactory = null;
  Queue testQueue = null;
  try {
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,
     "org.jnp.interfaces.NamingContextFactory");
   properties.put(Context.PROVIDER_URL, "jnp://127.0.0.1:1099");
   InitialContext jndiContext = new InitialContext(properties);
   queueConnectionFactory = (QueueConnectionFactory) jndiContext
     .lookup("ConnectionFactory");
   testQueue = (Queue) jndiContext.lookup("queue/testQueue");
  } catch (NamingException nameEx) {
   System.out.println("Naming Exception: " + nameEx.toString());
  }
  System.out.println("Made queue");
  QueueConnection queueConnection = null;
  try {
   System.out.println("Trying to send message");
   queueConnection = queueConnectionFactory.createQueueConnection();
   QueueSession queueSession = queueConnection.createQueueSession(
     false, Session.AUTO_ACKNOWLEDGE);
   QueueSender queueSender = queueSession.createSender(testQueue);
   TextMessage textMessage = queueSession.createTextMessage();
   System.out.println("Send each message");
   for (int msgCount = 0; msgCount < messages.length; msgCount++) {
    textMessage.setText(messages[msgCount]);
    queueSender.send(textMessage);
    System.out.println("\tSending line " + msgCount + " : "
      + messages[msgCount]);
   }
   textMessage.setText("End of message");
   queueSender.send(textMessage);
   System.out.println("\tSending last line " + " : "
     + textMessage.getText());
   queueConnection.close();
   System.out.println("Sender closed");
  } catch (javax.jms.JMSException jmsEx) {
   System.out.println("JMS Exception: " + jmsEx.toString());
  } finally {
   if (queueConnection != null) {
    try {
     queueConnection.close();
    } catch (javax.jms.JMSException jmse) {
     System.out.println("JMS exception.");
    }
   }
  }
 }
}
```

Listing 1.19: Konsument wiadomości – komunikacja typu point-to-point

```
package pl.mkalewski.jms;
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;
public class Consumer {
 public static void main(String[] args) {
  QueueConnectionFactory queueConnectionFactory = null;
  Queue testQueue = null;
  try {
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,
     "org.jnp.interfaces.NamingContextFactory");
   properties.put(Context.PROVIDER_URL, "jnp://127.0.0.1:1099");
   InitialContext jndiContext = new InitialContext(properties);
   queueConnectionFactory = (QueueConnectionFactory) jndiContext
     .lookup("ConnectionFactory");
   testQueue = (Queue) jndiContext.lookup("queue/testQueue");
  } catch (NamingException nameEx) {
   System.out.println("Naming Exception: " + nameEx.toString());
  }
  QueueConnection queueConnection = null;
  try {
   queueConnection = queueConnectionFactory.createQueueConnection();
   QueueSession queueSession = queueConnection.createQueueSession(
     false, Session.AUTO_ACKNOWLEDGE);
   QueueReceiver queueReceiver = queueSession
     .createReceiver(testQueue);
   queueConnection.start();
   TextMessage textMessage = null;
   while (true) {
    textMessage = (TextMessage) queueReceiver.receiveNoWait();
    System.out.println("\tReceiving line " + " : "
      + textMessage.getText());
    if (textMessage.getText().equals("End of message")) {
     break;
    }
   }
   queueConnection.close();
   System.out.println("Receiver closed");
  } catch (javax.jms.JMSException jmsEx) {
   System.out.println("JMS Exception: " + jmsEx.toString());
  } finally {
   if (queueConnection != null) {
    try {
     queueConnection.close();
    } catch (javax.jms.JMSException jmse) {
    }
   }
  }
 }
}
```

Listing 1.20: Producent wiadomości – komunikacja typu publish/subscribe

```
package pl.mkalewski.jms;
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;
public class Producer {
 public static void main(String[] args) {
  String messages[] = { "MESSAGE A", "MESSAGE B", "MESSAGE C",
    "MESSAGE D" };
  TopicConnectionFactory topicConnectionFactory = null;
  Topic testTopic = null;
  try {
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,
     "org.jnp.interfaces.NamingContextFactory");
   properties.put(Context.PROVIDER_URL, "jnp://127.0.0.1:1099");
   InitialContext jndiContext = new InitialContext(properties);
   topicConnectionFactory = (TopicConnectionFactory) jndiContext
     .lookup("ConnectionFactory");
   testTopic = (Topic) jndiContext.lookup("topic/testTopic");
  } catch (NamingException nameEx) {
   System.out.println("Naming Exception: " + nameEx.toString());
  }
  System.out.println("Made topic");
  TopicConnection topicConnection = null;
  try {
   System.out.println("Trying to send message");
   topicConnection = topicConnectionFactory.createTopicConnection();
   TopicSession topicSession = topicConnection.createTopicSession(
     false, Session.AUTO_ACKNOWLEDGE);
   TopicPublisher topicPublisher = topicSession
     .createPublisher(testTopic);
   TextMessage textMessage = topicSession.createTextMessage();
   System.out.println("Send each message");
   for (int msgCount = 0; msgCount < messages.length; msgCount++) {
    textMessage.setText(messages[msgCount]);
    topicPublisher.publish(textMessage);
    System.out.println("\tSending line " + msgCount + " : "
      + messages[msgCount]);
   }
   textMessage.setText("End of message");
   topicPublisher.publish(textMessage);
   System.out.println("\tSending last line " + " : "
     + textMessage.getText());
   topicConnection.close();
   System.out.println("Sender closed");
  } catch (javax.jms.JMSException jmsEx) {
   System.out.println("JMS Exception: " + jmsEx.toString());
  } finally {
   if (topicConnection != null) {
    try {
     topicConnection.close();
    } catch (javax.jms.JMSException jmse) {
```

```
      System.out.println("JMS exception.");
54    }
     }
56  }
 }
58 }
```

Listing 1.21: Konsument wiadomości – komunikacja typu publish/subscribe

```
package pl.mkalewski.jms;
2 import javax.jms.*;
import javax.naming.*;
4 import java.util.Properties;
public class Consumer {
6  static int foo;
 public static class ExListener implements MessageListener {
8   public void onMessage(Message msg) {
    foo = 1;
10  TextMessage tm = (TextMessage) msg;
    try {
12   System.out.println("\tReceiving line " + " : " + tm.getText());
    } catch (Throwable t) {
14   t.printStackTrace();
    }
16  }
 }
18 public static void main(String[] args) {
  TopicConnectionFactory topicConnectionFactory = null;
20  Topic testTopic = null;
  try {
22   Properties properties = new Properties();
    properties.put(Context.INITIAL_CONTEXT_FACTORY,
24    "org.jnp.interfaces.NamingContextFactory");
    properties.put(Context.PROVIDER_URL, "jnp://127.0.0.1:1099");
26  InitialContext jndiContext = new InitialContext(properties);
    topicConnectionFactory = (TopicConnectionFactory) jndiContext
28    .lookup("ConnectionFactory");
    testTopic = (Topic) jndiContext.lookup("topic/testTopic");
30  } catch (NamingException nameEx) {
   System.out.println("Naming Exception: " + nameEx.toString());
32  }
  System.out.println("Made topic");
34  TopicConnection topicConnection = null;
  try {
36   topicConnection = topicConnectionFactory.createTopicConnection();
    TopicSession topicSession = topicConnection.createTopicSession(
38    false, Session.AUTO_ACKNOWLEDGE);
    TopicSubscriber topicSubscriber = topicSession
40    .createSubscriber(testTopic);
    topicSubscriber.setMessageListener(new ExListener());
42  topicConnection.start();
    foo = 0;
44  System.out.println("waiting...");
    while (foo == 0)
```

```
46      ;
        topicConnection.close();
48      System.out.println("Receiver closed");
      } catch (javax.jms.JMSException jmsEx) {
50      System.out.println("JMS Exception: " + jmsEx.toString());
      } finally {
52      if (topicConnection != null) {
        try {
54        topicConnection.close();
        } catch (javax.jms.JMSException jmse) {
56      }
      }
58    }
  }
60}
```

1.5 Pozostałe zagadnienia

1. Wykorzystanie `Eclipse IDE` do implementacji EJB – http://trailblazer.demo.jboss.com/IDETrail/
2. Implementacja EJB 3.0 z wykorzystaniem `Eclipse IDE` – http://trailblazer.demo.jboss.com/EJB3Trail/
3. Klaster serwera JBoss.

Rozdział 2

Zadania

2.1 Zadanie nr 1 – *Uniform Reliable Broadcast*

Zaprojektuj i zaimplementuj z wykorzystaniem EJB aplikację rozproszoną rozgłaszania (ang. *broadcast*) wiadomości posiadającą następujące właściwości (zakładając poprawność większości procesów):

- jeśli poprawny proces p_i rozgłasza wiadomość m, to p_i ostatecznie dostarcza m;
- żadna wiadomość nie jest duplikowana;
- jeśli wiadomość m jest dostarczana przez pewien proces p_j, to wiadomość ta była uprzednio rozgłoszona przez pewien proces p_i;
- jeśli wiadomość m jest dostarczana przez pewien proces p_i (poprawny lub nie), to m będzie ostatecznie dostarczona przez każdy inny poprawny proces p_j.

2.2 Zadanie nr 2 – *Reliable Casual Broadcast*

Zaprojektuj i zaimplementuj z wykorzystaniem EJB aplikację rozproszoną rozgłaszania (ang. *broadcast*) wiadomości posiadającą następujące właściwości (zakładając poprawność większości procesów):

- jeśli poprawny proces p_i rozgłasza wiadomość m, to p_i ostatecznie dostarcza m;
- żadna wiadomość nie jest duplikowana;
- jeśli wiadomość m jest dostarczana przez pewien proces p_j, to wiadomość ta była uprzednio rozgłoszona przez pewien proces p_i;
- żaden proces p_i nie dostarcza wiadomości m_2 dopóki p_i nie dostarczy każdej wiadomości m_1, dla której zachodzi $m_1 \rightarrow m_2$.

2.3 Zadanie nr 3 – *Abortable Consensus*

Zaprojektuj i zaimplementuj z wykorzystaniem EJB aplikację rozproszonego uzgadniania (ang. *consensus*) posiadającą następujące właściwości:

- każdy poprawny proces, który zgłasza propozycję ostatecznie podejmuje decyzję lub przerywa procedurę (ang. *abort*);
- jeśli pojedynczy proces zgłasza propozycję nieskończenie wiele razy to ostatecznie podejmuje decyzję;

- żadne dwa procesy nie podejmują różnych decyzji;
- każda zdecydowana wartość musiała być zaproponowana przez jakiś proces.

Rozdział 3

TLA+

Rozdział 4

Lotus Domino

4.1 Przygotowanie środowiska

1. Instalacja środowiska Lotus Domino: Lotus Notes, Lotus Designer, Lotus Administrator.
2. Utworzenie kont dla użytkowników oraz wygenerowanie certyfikatów.
3. Konfiguracja oprogramowania do pracy w środowisku Domino.
4. Weryfikacja poprawności działania środowiska oraz uruchomienie aplikacji na serwerze.
5. Zachowanie własnych certyfikatów (c:/Lotus/Notes/Data/user.id lub c:/Lotus/Notes/Data/ids/people/*.id).

4.2 Praca grupowa w środowisku **Domino**

1. Wysyłanie wiadomości:
   - różne opcje dostarczenia (śledzenie drogi wiadomości);
   - priorytety wiadomości;
   - potwierdzenia otwarcia wiadomości;
   - zabranianie kopiowania wiadomości;
   - symbole wiadomości;
   - zaawansowane opcje dostarczania poczty (termin udzielenia odpowiedzi, przekierowanie odpowiedzi do innej osoby).
2. Czynności do wykonania (ang. *to do list*):
   - tworzenie i wysyłanie wniosków o zadania do wykonania;
   - opcje dostarczania (raport o dostarczeniu, priorytet, zabranianie składania przeciwnych propozycji i delegowania);
   - zmiana terminu, anulowanie i potwierdzanie wniosku o zadanie do wykonania;
   - śledzenie odpowiedzi na wniosek o zadanie do wykonania;
   - odpowiadanie na propozycję zmiany wniosku o zadanie do wykonania.
3. Delegowanie dostępu do kalendarza i zadań do wykonania:
   - określanie praw odczytu kalendarza i zadań do wykonania.
4. Ustawianie "czasu wolnego".

5. Tworzenie i wysyłanie zaproszeń na spotkania:
   - rezerwacja zasobów;
   - opcje dostarczenia (raport o dostarczeniu, priorytet, zabranianie składania przeciwnych propozycji i delegowania)
   - zmiana terminu, anulowanie i potwierdzanie zaproszenia na spotkanie;
   - śledzenie odpowiedzi na zaproszenie na spotkanie;
   - odpowiadanie na propozycję zmiany zaproszenia na spotkanie;
6. Wyszukiwanie wolnego czasu.
7. Rezerwowanie pomieszczeń i zasobów.
8. Tworzenie własnych zasobów.
9. Tworzenie agentów automatycznej obsługi dokumentów.
10. Zadanie: utworzyć własny zasób (salę) oraz zorganizować spotkanie w tej sali, na które należy wysłać zaproszenie ("zaproszenie na spotkanie") administratorowi w jego wolnym czasie.

4.3 Tworzenie aplikacji przy pomocy Lotus Designer

1. Formularze: typu dokument i odpowiedź, przekazywanie danych pomiędzy dokumentami, dodawanie pół różnych typów (w tym okien dialogowych z widoków – *use view dialog for choice*), przyciski akcji.
2. Widoki: selekcja dokumentów do widoku, tworzenie kolumn, przyciski akcji, modyfikacje pól wskazanych dokumentów jako akcje (*by form used* and *by field* oraz akcja *simple action*, *modify field*).
3. Struktury.
4. Strony.
5. Zestawy ramek.
6. Język formuł (m.in.: `@Command([FileSaveNewVersion])`; `@Command([FileCloseWindow])`; `@Author`; `@Today`; `@MailSend`; `@Created`; `@If(con1; act1; con2; act2;...; con99; act99; else)`).
7. Lotus Script:

```
Dim session As New NotesSession
Dim db As NotesDatabase
Dim dc As NotesDocumentCollection
Dim doc As NotesDocument
Dim subject As Variant
Set db = session.CurrentDatabase
Set dc = db.AllDocuments
Set doc = dc.GetFirstDocument
While Not(doc Is Nothing)
      subject = doc.GetItemValue(‘‘txtTemat’’)
      MessageBox subject(0)
      Set doc = dc.GetNextDocument(doc)
Wend
```