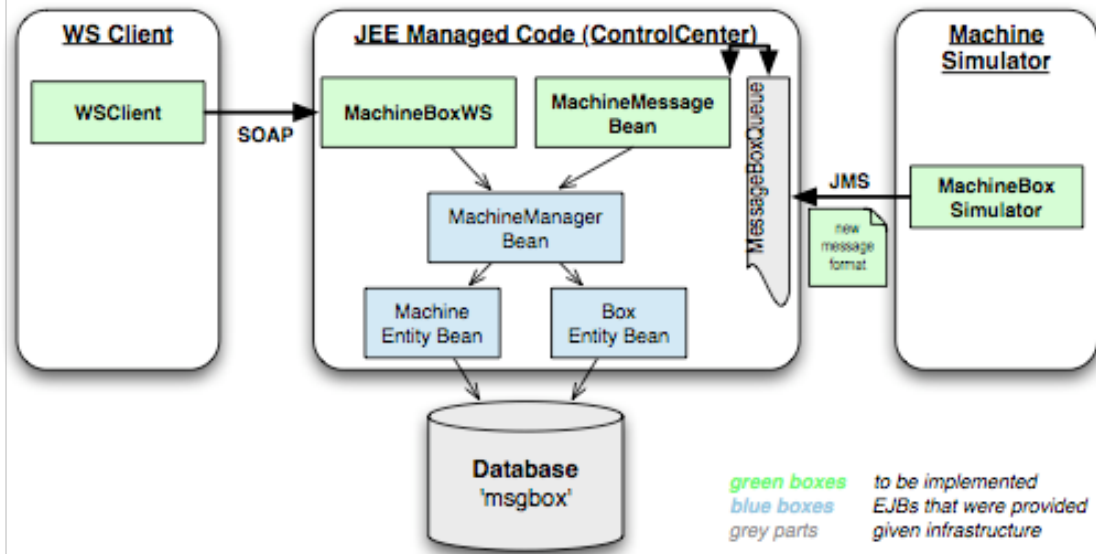


COMP53248 - Assignment 3 - Example Solution

The Big Picture

The system consists of three main parts:



1. The Machine Simulator
which is sending simulated machine events as JMS messages to the server
2. The ControlCenter as managed JEE server
which contains the main system logic in form of several Java Enterprise beans:
 - a) **session beans** (stateless) as service controllers (MachineManager)
 - b) **entity beans** (Box and Machine) encapsulating the persistent state
 - c) *to be done*: a **message-driven bean** to receive and act on JMS messages
 - d) *to be done*: a **session bean** that implement the **web-service** functions
3. A Web Service front-end
that allows to retrieve status information from the ControlCenter

The Assignment Tasks

1. extending the JMS simulator to send machine events to the JMS input queue of the managed server. Note:
 - the simulator should NOT directly interact with any managed bean but just deliver messages into the JMS queue
 - the simulator should also not read its own messages and then react on them, but rather the reaction to the JMS messages should be programmed inside the managed JEE server (see below)
2. Implementation of a Message-Driven Bean (MDB)
The managed sever must be extended with a so-called 'message-driven bean' that receives JMS events inside(!) the JEE server, interprets the content and invokes the MachineManager
3. Implementation of a Web-Service bean
inside the JEE server to publish some(!) functionality of the Machine Controller. Part of this task is to design a suitable WS interface
4. Implementation of a Web Service client
that contacts the Machine Control Server via its WS interface and allows some status checks etc.

Task 1: Machine Simulator

In order to extend the Machine Simulator correctly, you had to do two things:

a) Definition of a Message Format

The simplest solution is based on our example code, using a simple TextMessage format. One has to distinguish two different types of events and serialise each event into a straight-forward space-separated string format:

I. BoxStateEvents (B)

'B' SPACE MachineID SPACE BoxID SPACE When SPACE StateString

II. MachineStateEvents (M)

'M' SPACE MachineID SPACE StateString

MachineID and BoxID are longs, When is a Date and the StateStrings are textual encodings of the machine and box states.

A more sophisticated solution would use at least MapMessages to have typed messages in form of (key,value) pairs rather than the simple serialised strings; the optimal solution is though to introduce own Message classes for each individual message type that are extensions of ObjectMessages.

b) Implementation of a JMS Sender

The following code extends the MapMessage format used by the example code.

Project: 'messagebox-jms'

Source: 'MessageBoxSimulator.java'

```
package messagebox.test;

import java.util.Date;
import java.util.HashMap;
import java.util.Random;
import java.util.Properties;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSEException;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.QueueSession;
import javax.jms.MapMessage;
import javax.naming.InitialContext;
import messagebox.controller.DefaultBoxState;
import messagebox.controller.MachineManager;
import messagebox.controller.MachineState;
import messagebox.controller.BoxState.AuxState;
import messagebox.controller.BoxState.PrimaryState;
import messagebox.exception.BoxNotFoundException;
import messagebox.exception.ExistingMachineException;

public class MessageBoxSimulator
{
    private Long[] machineList = new Long[1];
    private HashMap<Long, Integer> machineCapacities = new HashMap<Long, Integer>();
    private Random rand = new Random();

    // initialise the expected state of the system (four machines with given IDs)
    public MessageBoxSimulator () {
        machineCapacities.put(201L, 7);
        machineCapacities.put(202L, 11);
        machineCapacities.put(203L, 13);
        machineCapacities.put(204L, 5);
        machineList = machineCapacities.keySet().toArray(machineList);
    }

    public void createRandomEvent( MapMessage msg ) throws JMSEException {
```

```

switch(rand.nextInt(2)) {
    case 0:
        msg.setChar("type", 'B');
        createBoxEvent(msg);
        break;
    case 1:
        msg.setChar("type", 'M');
        createMachineEvent(msg);
        break;
}
}

public void createBoxEvent( MapMessage msg ) throws JMSEException {
    //select a random machine
    Long macId = machineList[rand.nextInt(machineList.length)];
    msg.setLong("machineId", macId);

    //select a random box
    msg.setLong("boxId", rand.nextInt(machineCapacities.get(macId).intValue()) + 1);

    // set current date
    Date now = new Date();
    msg.setLong("date", now.getTime());

    //random primary state
    PrimaryState pStat = PrimaryState.NO_CHANGE;
    switch(rand.nextInt(3)){
        case 0:
            pStat = PrimaryState.FREE;
            break;
        case 1:
            pStat = PrimaryState.OCCUPIED;
            break;
        case 2:
            pStat = PrimaryState.NO_CHANGE;
            break;
    }
    msg.setString("primaryState", pStat.toString());

    //random aux state
    AuxState aStat = null;
    switch(rand.nextInt(3)){
        case 0:
            if(rand.nextBoolean()){
                aStat = AuxState.ALARM;
            }else{
                aStat = AuxState.CLEAR_ALARM;
            }
            break;
        case 1:
            if(rand.nextBoolean()){
                aStat = AuxState.DIRTY;
            }else{
                aStat = AuxState.CLEAR_DIRTY;
            }
            break;
        case 2:
            if(rand.nextBoolean()){
                aStat = AuxState.DEFECT;
            }else{
                aStat = AuxState.CLEAR_DEFECT;
            }
            break;
    }
    msg.setString("auxState", aStat.toString());
}

public void createMachineEvent(MapMessage msg ) throws JMSEException {
    //select a random machine
    msg.setLong("machineId", machineList[rand.nextInt(machineList.length)]);
}

```

```

//random machine state
MachineState mStat = null;
switch(rand.nextInt(2)){
    case 0:
        mStat = MachineState.OPERATIONAL;
        break;
    case 1:
        mStat = MachineState.OUT_OF_ORDER;
        break;
}
msg.setString("machineState", mStat.toString());
}

private static InitialContext getInitialContext() throws Exception{

    Properties props = new Properties();
    props.setProperty("java.naming.factory.initial",
        "com.sun.enterprise.naming.SerialInitContextFactory");
    props.setProperty("java.naming.factory.url.pkgs",
        "com.sun.enterprise.naming");
    props.setProperty("java.naming.factory.state",
        "com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl");

    // optional. Defaults to localhost. Only needed if web server is running
    // on a different host than the appserver
    props.setProperty("org.omg.CORBA.ORBInitialHost", "localhost");

    // optional. Defaults to 3700. Only needed if target orb port is not 3700.
    props.setProperty("org.omg.CORBA.ORBInitialPort", "3700");

    return new InitialContext(props);
}

private static QueueConnectionFactory getConnectionFactory() throws Exception{
    InitialContext ic = getInitialContext();
    return (QueueConnectionFactory)ic.lookup("jms/MessageBoxConnectionFactory");
}

private static Queue getQueue() throws Exception{
    InitialContext ic = getInitialContext();
    return (Queue)ic.lookup("jms/MessageBoxQueue");
}

public static void main(String[] args) throws Exception {

    // prepare event simulator
    MessageBoxSimulator sim = new MessageBoxSimulator();

    //create JMS connection
    QueueConnectionFactory cxn = getConnectionFactory();
    Destination dest = getQueue();
    QueueConnection conn = cxn.createQueueConnection();
    QueueSession sess = conn.createQueueSession(false,QueueSession.AUTO_ACKNOWLEDGE);

    try {
        MessageProducer prod = sess.createProducer(dest);
        MapMessage message = sess.createMapMessage();

        java.io.BufferedReader console =
            new java.io.BufferedReader(new java.io.InputStreamReader(System.in));
        String answer = " ";
        while (!(answer.charAt(0) == 'q' || (answer.charAt(0) == 'Q')) {

            sim.createRandomEvent(message);

            System.out.println("Next event: " + message.toString());
            System.out.println("Send (y/n/q) ?");
            try {

```

```

        answer = console.readLine();
        if ( answer.charAt(0) == 'y' ) {
            prod.send(message);
        }
    } catch (java.io.IOException e) {
        System.err.println("I/O exception: " + e.toString());
    }
}
}finally{
    try { conn.close(); } catch (JMSEException e) {
        System.out.println(e);
    }
}
System.out.println("goodbye.");
}
}
}

```

Task 2: Message-Driven Bean

It is important to note that the receiver of the machine and box events is a managed bean within the JEE server. The code itself is relatively simple: We basically only have to implement one method, the `onMessage()` function of a message driven bean, in which we parse the incoming message according to our defined format and then call the corresponding methods on the MachineManager bean. This method is linked via a few JEE annotations at the beginning of that file to any incoming message on the input queue.

a) Deployment (repeat)

Note that the following code assumes that you have configured the local JEE server correctly as specified on the handout of tutorial 9:

- there is a 'msgbox' Derby database with the expected schema and content and the server's DerbyPool uses this 'msgbox' database (DatabaseName)
- the server's JMS service is configured correctly:
 - there is a JMS queue 'MessageBoxQueue' which is linked under the JNDI entry 'jms/MessageBoxQueue'
 - there is a JMS connection factory 'jms/MessageBoxConnectionFactory'
 - if you use the servers in the SIT labs, the 'default_JMS_host' entry is set to include the 'ug.cs.usyd.edu.au' suffix

b) JMS Service Implementation

Project: 'messagebox-ejb'

Source: 'src/messagebox/jms/MachineMessageBean.java'

```

package messagebox.jms;

import java.lang.reflect.Field;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

import messagebox.controller.BoxState.AuxState;
import messagebox.controller.BoxState.PrimaryState;

/**
 * Message-Driven Bean implementation class for: MachineMessageBean
 *
 */

```

```

@MessageDriven(mappedName = "jms/MessageBoxQueue",
    activationConfig = { @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Queue"
    ) }
)
public class MachineMessageBean implements MessageListener {

    // helpful to get some debug messages out of the server
    static final Logger logger = Logger.getLogger("MachineMessageBean");

    Map<String, MachineState> macKeys = new HashMap<String, MachineState>();
    Map<String, PrimaryState> priKeys = new HashMap<String, PrimaryState>();
    Map<String, AuxState> auxKeys = new HashMap<String, AuxState>();

    @EJB
    MachineManager man;
    /**
     * Default constructor. Most code here is just to get some utility
     * data structures initialised so that we can decode the machine and
     * box states correctly...
     */
    public MachineMessageBean() {

        Field[] stats = MachineState.class.getFields();
        for(Field stat: stats){
            try{
                MachineState mstat = (MachineState)stat.get(null);
                macKeys.put(mstat.toString(), mstat);
            }catch(Exception ex){}
        }

        stats = PrimaryState.class.getFields();
        for(Field stat: stats){
            try{
                PrimaryState pstat = (PrimaryState)stat.get(null);
                priKeys.put(pstat.toString(), pstat);
            }catch(Exception ex){}
        }

        stats = AuxState.class.getFields();
        for(Field stat: stats){
            try{
                AuxState astat = (AuxState)stat.get(null);
                auxKeys.put(astat.toString(), astat);
            }catch(Exception ex){}
        }
    }

    /**
     * @see MessageListener#onMessage(Message)
     */
    public void onMessage(Message message) {
        TextMessage msg = null;
        try {
            if (message instanceof MapMessage) {
                MapMessage msg = (MapMessage) message;
                logger.info("MESSAGE BEAN: Message received: " + msg.toString());
                if ( msg.getChar("type") == 'B' ) {
                    Long macId = msg.getLong("machineId");
                    Long boxId = msg.getLong("boxId");
                    Date when = new Date (msg.getLong("date"));
                    DefaultBoxState bs = new DefaultBoxState();
                    bs.setPrimaryState(priKeys.get(msg.getString("primaryState")));
                    bs.addAuxState(auxKeys.get(msg.getString("auxState")));
                    man.setBoxState(macId, boxId, bs, when, "");
                }
                else if ( msg.getChar("type") == 'M' ) {
                    Long macId = msg.getLong("machineId");
                    MachineState mstat = macKeys.get(msg.getString("machineState"));
                    man.setMachineState(macId, mstat);
                }
            }
        }
    }
}

```

```

        }
    }
} catch (JMSEException e) {
    //System.out.println("JMSEException in onMessage(): " + e.toString());
} catch (Throwable t) {
    //System.out.println("Exception in onMessage(): " + t.getMessage());
}
}
}
}

```

Task 3: WebService Bean

Similar to the JMS machine simulator, we can distinguish here two sub-steps: Firstly we have to define which functions we want to publish as a web service; Secondly, we have to provide a corresponding implementation.

a) Web Service Design

You have to decide which operations to make available for WS clients.

We decided to just publish status check methods, but no state manipulation methods such as createMachine(), setBoxState() or setMachineState() all which are actually used only from the JMS/Machine side. Our WebService hence publishes the following six methods:

```

String  getMachineAddress( long machineId )
Integer getCapacity      ( long machineId )
String  getType          ( long machineId )
String  getMachineState  ( long machineId )
void    resetBoxStates   ( long machineId )
String  getBoxState      ( long machineId, long boxId )

```

An inherent problem is how to publish structured types such as the Address or the BoxState. We decided to use again a string version, so instead of MachineState, BoxState or Adress, we just return plain strings.

b) Web Service Implementation

A web service implementation in JEE5 is very easy: Bascially we implement a stateless session bean as facade before the actual controller bean and annotate it as **@WebService()**, as well as all public methods, which should be available to clients, as **@WebMethod**. That's it. The rest does JEE during deployment!

Project: 'messagebox-ejb'

Source: 'src/messagebox/ws/MessageBoxWS.java'

```

package messagebox.ws;

import javax.annotation.Resource;
import javax.ejb.EJB;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;

import java.util.List;

import messagebox.controller.MachineManager;
import messagebox.controller.BoxState;
import messagebox.controller.Address;

@Stateless
@WebService()
public class MessageBoxWS
{
    @EJB MachineManager man;
    private MachineManager getManager(){
        return man;
    }
}

```

```

@WebMethod
public String getMachineAddress(Long machineId){
    try{

        Address addr = getManager().getMachineAddress(machineId);

        return addrToString(addr);
    }catch(Exception nfe){
        return "No machine with ID of " + machineId + ": " + nfe.getMessage();
    }
}
private String addrToString(Address addr){
    StringBuffer buff = new StringBuffer();
    buff.append(" number:"); buff.append(addr.getNumber());buff.append('\n');
    buff.append(" street:"); buff.append(addr.getStreet());buff.append('\n');
    buff.append(" city:"); buff.append(addr.getCity());buff.append('\n');
    buff.append(" region:"); buff.append(addr.getRegion());buff.append('\n');
    buff.append(" zip:"); buff.append(addr.getZip());buff.append('\n');
    buff.append(" country:"); buff.append(addr.getCountry());buff.append('\n');
    return buff.toString();
}

@WebMethod
public Integer getCapacity(Long machineId){
    try{
        return getManager().getCapacity(machineId);
    }catch(Exception ex){
        return -1;
    }
}

@WebMethod
public String getType(Long machineId){
    try{
        return getManager().getType(machineId);
    }catch(Exception ex){
        return "unknown machine";
    }
}

@WebMethod
public String getMachineState(Long machineId){
    try{
        return getManager().getMachineState(machineId).toString();
    }catch(Exception ex){
        return "unknown machine";
    }
}

@WebMethod
public String getBoxState(Long machineId, Long boxId){
    try{
        String state = getManager().getBoxState(machineId,boxId).getPrimaryState().toString();
        List<BoxState.AuxState> auxS=getManager().getBoxState(machineId, boxId).getAuxState();
        for (int i=0; i<auxS.size(); i++)
            state += " " + ((BoxState.AuxState) auxS.get(i)).toString();
        return state;
    }catch(Exception ex){
        return "unknown machine or box";
    }
}

@WebMethod
public void resetBoxStates(Long machineId){
    try{
        getManager().resetBoxStates(machineId);
    }catch(Exception ex){
        ;
    }
}

```

```
}  
}
```

c) Web Service Deployment

The deployment of the web service is very simple, as we only have to include the file `MessageBoxService.java` in the server source, and Netbeans does then automatically generate the required Web Service endpoints and WSDL files.

This is one of the strengths of this managed server approach where a log of the low-level work such as WSDL generation or type marshalling is done automatically by the environment.

Task 4: Web-Service Client

As last step, we also have to implement a web service client to use our new service. You can do so as a JSP page, or as a stand-alone Java client. The following is an example for the later:

Project: 'messagebox-wsclient' (new)

Source: 'src/MsgBoxWSClient.java'

```
package messagebox.wsclient;  
  
import java.net.URL;  
import javax.xml.namespace.QName;  
import messagebox.wsclient.MessageBoxWS;  
import messagebox.wsclient.MessageBoxWSService;  
  
public class MsgBoxWSClient {  
  
    // @WebServiceRef(wsdlLocation="http://localhost:8080/MessageBoxWSService/MessageBoxWS?wsdl")  
    MessageBoxWSService service;  
  
    public static void main(String[] args) {  
        Long macId = 201L;  
        String host = "localhost";  
        String port = "8080";  
  
        try {  
  
            if ( args.length > 1 ) host = args[1];  
            if ( args.length > 2 ) host = args[2];  
  
            if (args.length > 0) {  
                macId = Long.parseLong(args[0]);  
            } else {  
                System.out.println("Please enter a machine id: ");  
                java.io.BufferedReader console = new  
                java.io.BufferedReader(new java.io.InputStreamReader(System.in));  
                String answer = console.readLine();  
                macId = Long.parseLong(answer);  
            }  
  
            MsgBoxWSClient client = new MsgBoxWSClient();  
            client.getService(host, port);  
            client.doTest(macId);  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
  
    public void getService(String host, String port) throws Exception {  
        service = new MessageBoxWSService(  
            new URL("http://" + host + ":" + port + "/MessageBoxWSService/  
            MessageBoxWS?wsdl"),
```

```

        new QName("http://ws.messagebox/", "MessageBoxWSService")
    );
}

public void doTest(Long machineId) {
    try {
        System.out.println("Retrieving the port for service: "+service);
        MessageBoxWS port = service.getMessageBoxWSPort();

        System.out.println("Status of Machine "+machineId);
        String response = port.getType(machineId);
        System.out.println("Type:      " + response);
        Integer capacity = port.getCapacity(machineId);
        System.out.println("Capacity: " + capacity);
        String status = port.getMachineState(machineId);
        System.out.println("Status:   " + status);
        for ( long i=1; i<=capacity; i++ ) {
            String state = port.getBoxState(machineId, i);
            System.out.println("  box " + i + ": " + state);
        }

    } catch(Exception e) {
        e.printStackTrace();
    }
}
}

```