APPENDIX A
SOFTWARE LISTINGS

1 Node Component

1.1 NodeEJB

```
////////////////////////////////////////////////////////////////////////////
/**
 * Name            NodeEJB
 * Description : The node component administers the conditional probability table,
 *               PI and LAMBDA of a Bayesian Network node
 *
 *
 *
 *
 * Change Control
 *
 */

////////////////////////////////////////////////////////////////////
// Imports
import javax.ejb.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.*;
import java.util.*;

////////////////////////////////////////////////////////////////////
/**
 * Implementation class of the NodeEJB entity bean.
 * Author Anet Potgieter (anet@bayesbean.com)
 * Version 1.0
 *
 * EJB
 *   Type:       Entity
 *   Description: NodeEJB Bean
 *   Home :      NodeHome
 *   Remote:     Node
 *   Persistence: Bean-managed
 *
 */

public class NodeEJB implements EntityBean {

    private String node;
    private String nodeName;
    private EntityContext context;
    private Connection con;
    private String dbName = "java:comp/env/jdbc/BabeDB";
    private int nrParentStates = 0;
    private int nrStates = 0;
    private Matrix cpm;
    private Matrix occurrences;
    private Array lambda = new Array(0);
    private Array piX = new Array(0);
```

```
  private Array evidence = new Array(0);
  private String [] incomingLinks = new String[0];
  private String [] outgoingLinks = new String[0];
  private String [] stateNames = new String [0];
  private int nrparents;
  private int nrchildren;

  private boolean trace = false;
  private boolean trace2 = false;

  int test = 1;

////////////////////////////////////////////////////////////////////////
// Business Routines
// -----------------

/**********************************************************************
 *  Name : reset
 *  Description : Reset pi, lambda and external pi for this node
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

  public void reset() {

  if (trace) {System.out.println("\n Resetting PI and Lambda for Node" + node);}

  for (int cnt = 0; cnt < piX.length();cnt++){
   piX.set(cnt,1.0) ;}

  for (int cnt = 0; cnt < lambda.length();cnt++){
   lambda.set(cnt,1.0) ;}

  for (int cnt = 0; cnt < evidence.length();cnt++){
   evidence.set(cnt,1.0) ;}

  }

/**********************************************************************
 *  Name : setNoEvidence
 *  Description : Set the evidence of the node to "no evidence"
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

public void setNoEvidence() {

  if (trace) {System.out.println("\n Setting node to No Evidence" + node);}

  for (int cnt = 0; cnt < evidence.length();cnt++){
    evidence.set(cnt,1.0) ;}
  }
```

```
/**********************************************************************
 *  Name : getNrStates
 *  Description : Get the number of states for this node
 *  Input parameters : none
 *  Return : nr of states
 **********************************************************************/

public int getNrStates() {
  return nrStates;
}

  /**********************************************************************
 *  Name : getPI
 *  Description : Get the PI of this node
 *  Input parameters : none
 *  Return : Array
 **********************************************************************/

  public Array getPI() {

 if (trace) {System.out.println("\nNode" + node + " getPI PI :");

 for (int cnt = 0; cnt < piX.length();cnt++){
    System.out.println(piX.get(cnt)+ ",");}

 System.out.println("\nEnd Node" + node + " PI");}
     return piX;
  }

/**********************************************************************
 *  Name : setPI
 *  Description : Set the PI of this node
 *  Input parameters : new PI
 *  Return : none
 **********************************************************************/


public void setPI(Array pi) {

 if (pi.length() <= this.piX.length()) {
   for (int cnt = 0; cnt < pi.length();cnt++){
    this.piX.set(cnt,pi.get(cnt));
   }

   if (pi.length() != this.piX.length()) {
      System.out.println("\n NodeEJB : Invalid length for setting PI for Node" + node);
   }
 }
 if (trace) {System.out.println("\nNode" + node + " setPI PI :");

 for (int cnt = 0; cnt < pi.length();cnt++){
   System.out.println(pi.get(cnt)+ ",");}
   System.out.println("\nNode" + node + " END setPI PI :");
```

```
  }
}

/**********************************************************************
 *  Name : getEvidence
 *  Description : Get the evidence of this node
 *  Input parameters : none
 *  Return : Array
 **********************************************************************/


public Array getEvidence() {
  if (trace) {System.out.println("\nNode" + node + " getEvidence Evidence :");

  for (int cnt = 0; cnt < evidence.length();cnt++){
    System.out.println(evidence.get(cnt)+ ",");}
    System.out.println("\nEnd Node" + node + " Evidence");
  }
  return evidence;
}

/**********************************************************************
 *  Name : setEvidence
 *  Description : Set the evidence of this node
 *  Input parameters : new Evidence
 *  Return : none
 **********************************************************************/

public void setEvidence(int parentState,int state,boolean learn) {



  this.evidence = new Array(nrStates,0.0);
  this.evidence.set(state - 1,1.0);

  if (learn){
    if (trace2) {System.out.println("\nNode" + node + " setEvidence Evidence :");

      for (int cnt = 0; cnt < nrStates;cnt++){
        System.out.println("\nCount:" + cnt + "EV : " + this.evidence.get(cnt)+ ",");
        System.out.println("\nCOUNT:" + cnt + "PI : " + cpm.get(parentState - 1,cnt)+ ",");
        System.out.println("\nCOUNT:" + cnt + "Occurences : " + occurrences.get(parentState - 1,cnt));
      }
      System.out.println("Node" + node + " END setEvidence:");
    }

    Array newProbs = new Array(nrStates,1.0);
    double occ = this.occurrences.get(parentState - 1,state - 1) + 1.0;
    this.occurrences.set(parentState - 1,state - 1,occ);

    for (int cnt = 0; cnt < newProbs.length(); cnt++){
      newProbs.set(cnt, this.occurrences.get(parentState - 1,cnt));
    }
```

```
    newProbs = newProbs.normalize();
    for (int cnt = 0; cnt < newProbs.length(); cnt++){
         this.cpm.set(parentState - 1,cnt,newProbs.get(cnt));
    }
  }

  if (trace2) {System.out.println("\nNode" + node + " setEvidence Evidence :");

    for (int cnt = 0; cnt < this.evidence.length();cnt++){
      System.out.println(this.evidence.get(cnt)+ ",");
    }
    System.out.println("\nNode" + node + " END setEvidence:");
  }
}

/**********************************************************************
 *  Name : setLAMBDA
 *  Description : Set the LAMBDA of this node
 *  Input parameters : new lambda
 *  Return : none
 **********************************************************************/

public void setLAMBDA(Array lambda) {
  this.lambda = lambda;
}

/**********************************************************************
 *  Name : setCPM
 *  Description : Set the CPM of this node
 *  Input parameters : new CPM
 *  Return : none
 **********************************************************************/

public void setCPM(Matrix CPM) {
  this.cpm = CPM;
}

/**********************************************************************
 *  Name : getCPM
 *  Description : Get the CPM of this node
 *  Input parameters : new lambda
 *  Return : none
 **********************************************************************/

  public Matrix getCPM() {
    return this.cpm;
}

/**********************************************************************
 *  Name : getLAMBDA
 *  Description : Get the LAMBDA of this node
 *  Input parameters : none
 *  Return : Array
```

```
 ***********************************************************************/

public Array getLAMBDA() {
  Array prod = this.lambda.termProduct(evidence);
  return prod;
}

/***********************************************************************
 *  Name : getBelief
 *  Description : Get the belief of this node
 *  Input parameters : none
 *  Return : Array
 ***********************************************************************/

public Array getBelief() {
  Array prod = piX.termProduct(lambda);
  prod = prod.termProduct(evidence);
  Array normProd = prod.normalize();
  return normProd;
}

/***********************************************************************
 *  Name : getStates
 *  Description : Get the state names of this node
 *  Input parameters : none
 *  Return : List of strings
 ***********************************************************************/

public String [] getStates () {
  return this.stateNames;
}

/***********************************************************************
 *  Name : getNode
 *  Description : Get the node name of this node
 *  Input parameters : none
 *  Return : String
 ***********************************************************************/

  public String getNode () {
    return this.nodeName;
}
/***********************************************************************
 *  Name : getOutgoingLinks
 *  Description : Get the outgoing links of this node
 *  Input parameters : none
 *  Return : List of strings
 ***********************************************************************/

  public String [] getOutgoingLinks () {
    return outgoingLinks;
}
```

```
/*********************************************************************
 *  Name : getIncomingLinks
 *  Description : Get the incoming links of this node
 *  Input parameters : none
 *  Return : List of strings
 *********************************************************************/

public String [] getIncomingLinks () {
  return incomingLinks;
}

/*********************************************************************
 *  Name : multiplyTransposeCPM
 *  Description : Multiply the transpose of the CPM with an array
 *  Input parameters : input Array
 *  Return : Array
 *********************************************************************/

public Array multiplyTransposeCPM(Array inputArray) {
  Array arr = new Array(nrStates,1.0);
  Matrix transcpm = cpm.transpose();
  arr = transcpm.matMult(inputArray);
  return arr;
}

/*********************************************************************
 *  Name : multiplyCPM
 *  Description : Multiply the CPM with an array
 *  Input parameters : input Array
 *  Return : Array
 *********************************************************************/

public Array multiplyCPM(Array inputArray) {
  Array arr = new Array(nrStates,1.0);
  arr = cpm.matMult(inputArray);
  return arr;
}

//////////////////////////////////////////////////////////////////////
// The following are required EJB methods, called by the container

/*********************************************************************************
 *  Name : ejbCreate
 *  Description : ejbCreate is called by the EJB container after setEntityContext
 *  Input parameters : Bayesian Network Node Name
 *  Return : String
 *********************************************************************************/

public String ejbCreate(String node)
  throws CreateException {

  try {
    if (trace) {System.out.println("\nNode" + node + " ejbCreate");}
```

```
    this.node = node;
    loadCPMDimensions(node);
    lambda = new Array(nrStates,1.0);
    piX = new Array(nrStates,1.0);
    evidence = new Array(nrStates,1.0);
    stateNames = getStateNames(nrStates);
    nodeName = getNodeName();
    loadCPMByNode(node);

    loadPL("node_pi", "pi", piX);
    loadPL("node_lambda", "lambda", lambda);
    loadPL("node_evidence", "evidence", evidence);

    if (trace) {System.out.println("\nNode (EJBCreate) " + node + " PI :");

    for (int cnt = 0; cnt < piX.length();cnt++){
      System.out.println(piX.get(cnt)+ ",");}

    System.out.println("\nEnd Node" + node + " PI");}

    outgoingLinks = getLinks("parent");
    incomingLinks = getLinks("child");
  } catch (Exception ex) {
    throw new EJBException("ejbCreate: " +
    ex.getMessage());
  }

  if (trace) {System.out.println("\nEnd Node" + node + " ejbCreate");}
  return node;
}

/*********************************************************************
 *  Name : ejbFindByPrimaryKey
 *  Description : Find the NodeEJB instance that matches the given primary key.
 *  Input parameters : Bayesian Network Node Name
 *  Return : String
 *********************************************************************/


public String ejbFindByPrimaryKey(String primaryKey) throws FinderException {

  boolean result;
  try {
     result = selectByPrimaryKey(primaryKey);
  } catch (Exception ex) {
    throw new EJBException("ejbFindByPrimaryKey: " +
    ex.getMessage());
  }
  if (result) {
    return primaryKey;
  }
  else {
```

```
    throw new ObjectNotFoundException("Row for id " + primaryKey + " not found.");
  }
}

/**********************************************************************
 *  Name : ejbRemove
 *  Description : Called by container before data removed from database.
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

public void ejbRemove() {
  if (trace) {System.out.println("\nNode" + node + " ejbRemove");}
  try {
  }
  catch (Exception ex) {
    throw new EJBException("ejbRemove: " +
    ex.getMessage());
   }
  if (trace) {System.out.println("\nEnd Node" + node + " ejbRemove");}
}

/**********************************************************************
 *  Name : ejbLoad
 *  Description : Called by container to refresh entity Bean's state.
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

public void ejbLoad() {
  if (trace) {System.out.println("\nNode" + node + " ejbLoad");}
  try {
    //loadCPMDimensions(node);
    loadCPMByNode(node);

    loadPL("node_pi", "pi", piX);
    loadPL("node_lambda", "lambda", lambda);
    loadPL("node_evidence", "evidence", evidence);

    if (trace) {System.out.println("\nNode" + node + " ejbLoad PI :");

    for (int cnt = 0; cnt < piX.length();cnt++){
      System.out.println(piX.get(cnt)+ ",");}

    System.out.println("\nEnd Node" + node + " ejbloadPI");}

  } catch (Exception ex) {
    throw new EJBException("ejbLoad: " +
    ex.getMessage());
  }
  if (trace) {System.out.println("\nend Node" + node + " ejbLoad");}
}
```

```
/*********************************************************************
 *  Name : ejbStore
 *  Description : save Bean's state to database.
 *  Input parameters : none
 *  Return : none
 *********************************************************************/

public void ejbStore() {
  if (trace) {System.out.println("\nNode" + node + " ejbStore");}
  try {
    storePL("node_pi", "pi", piX);
    storePL("node_lambda", "lambda", lambda);
    storePL("node_evidence", "evidence", evidence);
    updateCPMByNode();
  } catch (Exception ex) {
    throw new EJBException("ejbLoad: " +
    ex.getMessage());
  }
  if (trace) {System.out.println("\nend Node" + node + " ejbStore");}
 }

 /*********************************************************************
 *  Name : ejbPostCreate
 *  Description : Called by container after ejbCreate.
 *  Input parameters : none
 *  Return : none
 *********************************************************************/
 public void ejbPostCreate(String node) {}

 ////////////////////////////////////////////////////////////////////
 // The following are callback methods, called by the container to
 // notify the Bean that some event is about to occur.

 /*********************************************************************
 *  Name : setEntityContext
 *  Description : Called by container to set Bean context.
 *  Input parameters : context
 *  Return : none
 *********************************************************************/

 public void setEntityContext(EntityContext context) {
   if (trace) {System.out.println("\nNode" + node + " setEntityContext");}
   this.context = context;
   try {
     makeConnection();
   } catch (Exception ex) {
     throw new EJBException("Unable to connect to database. " +
     ex.getMessage());
   }
   if (trace) {System.out.println("\nend Node" + node + " setEntityContext");}
 }

/*********************************************************************
```

```
 *  Name : unsetEntityContext
 *  Description : Called by container to unset Bean context.
 *  Input parameters : context
 *  Return : none
 ********************************************************************/

 public void unsetEntityContext() {
  if (trace) {System.out.println("\nNode" + node + " unsetEntityContext");}
  try {
     con.close();
  } catch (SQLException ex) {
    throw new EJBException("unsetEntityContext: " + ex.getMessage());
  }
  if (trace) {System.out.println("\nend Node" + node + " unsetEntityContext");}
 }

 /********************************************************************
 *  Name : ejbActivate
 *  Description : Called by container before Bean swapped into memory.
 *  Input parameters : none
 *  Return : none
 ********************************************************************/

 public void ejbActivate() {
  if (trace) {System.out.println("\nNode" + node + " ejbActivate");}
  node = (String)context.getPrimaryKey();
  if (trace) {System.out.println("\nend Node" + node + " ejbActivate");}
 }

 /********************************************************************
 *  Name : ejbPassivate
 *  Description : Called by container before Bean swapped into storage.
 *  Input parameters : none
 *  Return : none
 ********************************************************************/

 public void ejbPassivate() {
  if (trace) {System.out.println("\nNode" + node + " ejbPassivate");}
  node = null;
  if (trace) {System.out.println("\nend Node" + node + " ejbPassivate");}
 }


////////////////////////////////////////////////////////////////////////////
/********************** Database Routines ************************/

 /********************************************************************
 *  Name : makeConnection
 *  Description : Connect to the database
 *  Input parameters : none
 *  Return : none
 ********************************************************************/
```

```
 private void makeConnection() throws NamingException, SQLException {
  if (trace) {System.out.println("\nNode" + node + " makeConnection");}
  InitialContext ic = new InitialContext();
  DataSource ds = (DataSource) ic.lookup(dbName);
  con =  ds.getConnection();
  if (trace) {System.out.println("\nend Node" + node + " makeConnection");}
 }

 /**********************************************************************
  *  Name : loadCPMDimensions
  *  Description : Determine dimensions of CPM
  *  Input parameters : Node Name
  *  Return : none
  **********************************************************************/

 private void loadCPMDimensions(String primaryKey) throws SQLException {

  // Determine nr states for this node
  String selectStatement = "select count(distinct state) " + "from cpm where node = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, primaryKey);

  ResultSet rs = prepStmt.executeQuery();

  if (rs.next()) {
    this.nrStates = rs.getInt(1);
    if (trace) {System.out.println("\nNodeBean " + node + " Nr of States = " + nrStates + "\n");}
    prepStmt.close();
  }
  else {
    prepStmt.close();
    throw new NoSuchEntityException("Could not determine nr of states for node " + primaryKey +
              " in database.");
  }
  // Determine Matrix indices - total nr of states for this node
  selectStatement = "select count(state) " + "from cpm where node = ? ";
  prepStmt = con.prepareStatement(selectStatement);
  prepStmt.setString(1, primaryKey);
  rs = prepStmt.executeQuery();

  if (rs.next()) {
    int totalNrStates = rs.getInt(1);
    this.nrParentStates = totalNrStates / nrStates;
    if (trace) {System.out.println("\nNodeBean " + node + "nrParentStates = " + nrParentStates +
"\n");}
    prepStmt.close();
  }
  else {
    prepStmt.close();
    throw new NoSuchEntityException("Could not determine total nr of states for node " + primaryKey
+
              " in database.");
```

```
  }

 }

 /*********************************************************************
  *  Name : loadCPMByNode
  *  Description : Retrieve CPM for this node
  *  Input parameters : Node Name
  *  Return : none
  *********************************************************************/

 private void loadCPMByNode(String primaryKey) throws SQLException {
  if (trace) {System.out.println("\nNode " + node + " loadCPMByNode");}
  String selectStatement = "select probability, state, parent_state, occurences " +
  "from cpm where node = ? order by parent_state,state";
  PreparedStatement prepStmt = con.prepareStatement(selectStatement);

  prepStmt.setString(1, primaryKey);
  ResultSet rs = prepStmt.executeQuery();

  double [][] vals = new double [nrParentStates][nrStates];
  double [][] history = new double [nrParentStates][nrStates];

  for (int i=0; i < nrParentStates; i++){
    for (int j = 0; j < nrStates; j++) {
      rs.next();
      double prob = rs.getDouble(1);

      if (trace) {System.out.println("\nload Node CPM" + node + " Prob = " + "State," +
      rs.getString(2) + "Parent_State," + rs.getString(3) + ", PROBABILITY = " + rs.getDouble(1)
+"\n");}
      vals[i][j] = prob;
      int nr = rs.getInt(4);
      history[i][j] = nr;
    }
  }

  prepStmt.close();
  cpm = new Matrix(vals);
  occurrences = new Matrix(history);

  if (trace) {System.out.println("\nLoad CPM Node" + node + " loadCPM PI :");

  for (int cnt = 0; cnt < piX.length();cnt++){
    System.out.println(piX.get(cnt)+ ",");}

  System.out.println("\nEnd Node" + node + " loadCPM PI ");}

  if (trace) {System.out.println("\nend Node loadCPMByNode");}
 }

 /*********************************************************************
  *  Name : updateCPMByNode
```

```
 *  Description : Update CPM with updatet probabilities
 *  Input parameters : New Probabilities
 *  Return : none
 ************************************************************************/

private void updateCPMByNode() throws SQLException {
  if (trace) {System.out.println("\nNode " + node + " updateCPMByNode");}
  String updateStatement = "update cpm set probability = ?,occurences = ? where " +
                           "node = ? and state = ? and parent_state = ?";

  if (trace) {System.out.println(updateStatement);}
  int rowCount;
  PreparedStatement prepStmt =
  con.prepareStatement(updateStatement);


  prepStmt.setString(3, this.node);
  if (trace) {System.out.println("\nAfter setstring Node = " + this.node);}

  for (int i=0; i < nrParentStates; i++){
    for (int j = 0; j < nrStates; j++) {

    if (trace) {System.out.println("\n Updating Node CPM " + node + " Parent State = " + (i + 1) +
    "," + " State = " + (j + 1) + ", PROBABILITY = " + cpm.get(i,j) +"\n");}
      prepStmt.setInt(4, j+1);
      if (trace) {System.out.println("\nAfter setInt state = " + j);}

      if (nrParentStates == 1) { // A Root
        prepStmt.setInt(5, i);
        if (trace) {System.out.println("\nAfter setInt state = " + i);}
      }
      else
      {
        prepStmt.setInt(5, i + 1);
        if (trace) {System.out.println("\nAfter setInt state = " + (i + 1));}
      }

      prepStmt.setDouble(1, cpm.get(i,j));
      prepStmt.setInt(2, (int)occurrences.get(i,j));
      if (trace) {System.out.println("\nold CPM [" + i + "][" + j + "] = " + cpm.get(i,j));}
      rowCount = prepStmt.executeUpdate();

      if (trace) {System.out.println("\n Node updated " + rowCount);}

    }
  }

  prepStmt.close();

  if (trace) {System.out.println("\nend Node updateCPMByNode");}
}

/**********************************************************************
```

```
 *  Name : getNodeName
 *  Description : Retrieve Node Name for Node
 *  Input parameters : none
 *  Return : String
 ************************************************************************/

private String getNodeName() throws SQLException {

  String selectStatement = "select description from nodes where node = ?";

  if (trace2){System.out.println("\ngetStateNames" + selectStatement);}

  PreparedStatement prepStmt = con.prepareStatement(selectStatement);

  prepStmt.setString(1, node);
  ResultSet rs = prepStmt.executeQuery();
  rs.next();
  String nodeName = rs.getString(1);
  prepStmt.close();

  return(nodeName);
}

/**********************************************************************
   *  Name : getStateNames
   *  Description : Retrieve State Names for Node
   *  Input parameters : none
   *  Return : State Names
   ************************************************************************/

  private String[] getStateNames(int stateCnt) throws SQLException {

    String [] states = new String [stateCnt];
    String selectStatement = "select state,description from states where node = ? order by state";

    if (trace){System.out.println("\ngetStateNames" + selectStatement);}

    PreparedStatement prepStmt = con.prepareStatement(selectStatement);

    prepStmt.setString(1, node);
    ResultSet rs = prepStmt.executeQuery();

    for (int i=0; i < stateCnt; i++){
      rs.next();
      String state = rs.getString(2);

      if (i < stateCnt) {
        if (trace) {System.out.println("\nNode " + node + "State " + i + " = " + state);}
        states[i] = state; }
      else {
      throw new NoSuchEntityException("GetStates ARRAY BOUND ERROR !!!!"); }
    }
```

```
    prepStmt.close();

    return(states);
  }

  /*********************************************************************
   *  Name : getLinks
   *  Description : Retrieve Incoming Links
   *  Input parameters : relation = "parent"/"child"
   *  Return : none
   *********************************************************************/

   private String[] getLinks(String relation) throws SQLException {

    int nr = 0;
    String [] links = new String [0];

    // Determine nr incoming links for this node
    String selectStatement = "select count(link) from links where " + relation + " = ?";
    PreparedStatement prepStmt =
    con.prepareStatement(selectStatement);
    prepStmt.setString(1, node);

    ResultSet rs = prepStmt.executeQuery();

    if (rs.next()) {
      nr = rs.getInt(1);
      prepStmt.close();
    }
    else {
      prepStmt.close();
      throw new NoSuchEntityException("Could not determine nr of links for " + relation + " for node "
+ node +
                " in database.");
    }

    if (nr == 0) {return links;}
    links = new String [nr];
    selectStatement = "select link from links where " + relation + " = ?";

    prepStmt = con.prepareStatement(selectStatement);

    prepStmt.setString(1, node);
    rs = prepStmt.executeQuery();

    for (int i=0; i < nr; i++){
      rs.next();
      String link = rs.getString(1);

      if (i < links.length) {
        links[i] = link; }
      else {
      throw new NoSuchEntityException("ARRAY BOUND ERROR !!!!"); }
```

```
    }

    prepStmt.close();

    return links;
}

/**********************************************************************
 *  Name : selectByPrimaryKey
 *  Description : Select node using primary key
 *  Input parameters : Node Name
 *  Return : boolean = status of retrieval
 **********************************************************************/

private boolean selectByPrimaryKey(String primaryKey) throws SQLException {

  String selectStatement =
      "select node " +
      "from nodes where node = ? ";
  PreparedStatement prepStmt =
      con.prepareStatement(selectStatement);
  prepStmt.setString(1, primaryKey);

  ResultSet rs = prepStmt.executeQuery();
  boolean result = rs.next();
  prepStmt.close();

  return result;
}

/**********************************************************************
 *  Name : insertPL
 *  Description : Update PI or LAMBDA or evidence
 *  Input parameters : Table Name, Values
 *  Return : none
 **********************************************************************/

private void insertPL (String TableName,Array PL) throws SQLException {

  String selectStatement =
      "select node " +
      "from " + TableName + " where node = ? ";
  PreparedStatement prepStmt =
      con.prepareStatement(selectStatement);

  prepStmt.setString(1, node);

  ResultSet rs = prepStmt.executeQuery();

  if (rs.next()) {
    prepStmt.close();
    deletePL(TableName);
  }
```

```
  else {
    prepStmt.close();

  }

  String insertStatement = "insert into " + TableName + " values ( ? , ? , ? )";

  prepStmt = con.prepareStatement(insertStatement);
  prepStmt.setString(1, node);
  for (int cnt = 1;cnt <= PL.length();cnt++) {
    prepStmt.setInt(2, cnt);
    prepStmt.setDouble(3, PL.get(cnt-1));
    prepStmt.executeUpdate();
  }
  prepStmt.close();

}

/**********************************************************************
 *  Name : deletePL
 *  Description : Delete PI/LAMBDA/Evidence
 *  Input parameters : Table Name
 *  Return : none
 **********************************************************************/

private void deletePL(String TableName) throws SQLException {

  String deleteStatement =
  "delete from " + TableName + " where node = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(deleteStatement);

  prepStmt.setString(1, this.node);
  prepStmt.executeUpdate();
  prepStmt.close();

}

/**********************************************************************
 *  Name : loadPL
 *  Description : Load PI/LAMBDA/Evidence
 *  Input parameters : Table Name, Field Name, Array
 *  Return : none
 **********************************************************************/

private void loadPL(String TableName, String Field, Array arr) throws SQLException {

  String selectStatement =
  "select node, idx, " + Field +
  " from " + TableName + " where node = ? order by node,idx";

  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
```

```
    prepStmt.setString(1, this.node);

    ResultSet rs = prepStmt.executeQuery();
    int cnt = 1;
    while (rs.next()) {
      if (cnt <= arr.length()) {
        arr.set(cnt - 1, rs.getDouble(3));
        cnt++;}
    }
    prepStmt.close();

    if (trace) {System.out.println("\nNode" + node + " loadPL " + Field );

    for (cnt = 0; cnt < piX.length();cnt++){
      System.out.println(piX.get(cnt)+ ",");}

    System.out.println("\nEnd Node" + node + Field );}

  }


  /**********************************************************************
   *  Name : storePL
   *  Description : store PI/LAMBDA/Evidence
   *  Input parameters : Table Name, Field Name, Array
   *  Return : none
   **********************************************************************/

  private void storePL(String TableName, String Field, Array arr) throws SQLException {
    int rowCount = 0;
    String updateStatement = "update " + TableName + " set " + Field + " =  ? " +
    "where node = ? and idx = ?";
    PreparedStatement prepStmt =
    con.prepareStatement(updateStatement);
    prepStmt.setString(2, this.node);
    for (int cnt = 1;cnt <= arr.length(); cnt++) {
      prepStmt.setInt(3, cnt);
      prepStmt.setDouble(1, arr.get(cnt - 1));
      rowCount = prepStmt.executeUpdate();
    }
    prepStmt.close();

    if (rowCount == 0) {
      throw new EJBException("Storing pi or lambda for node " + this.node + " failed.");
    }
  }
} // NodeEJB
```

1.2 Node Home Interface

```
//////////////////////////////////////////////////////////////////////
/**
 * Name         NodeHome
 * Description : HOME INTERFACE FOR THE NODE ENTITY BEAN CLASS
 *
 *
 *
 *
 * Change Control
 *
 */

//////////////////////////////////////////////////////////////////////

import java.rmi.RemoteException;
import javax.ejb.*;
import java.util.Collection;

public interface NodeHome extends EJBHome {
  public Node create(String node) throws CreateException, RemoteException;
  public Node findByPrimaryKey(String node) throws FinderException, RemoteException;
}
```

1.3 Node Remote Interface

```
////////////////////////////////////////////////////////////////////////
/**
 * Name         Node
 * Description : REMOTE INTERFACE FOR THE NODE ENTITY BEAN CLASS
 *
 *
 *
 *
 * Change Control
 *
 */

////////////////////////////////////////////////////////////////////////


import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Node extends EJBObject {
  public String getNode() throws RemoteException;
  public int getNrStates()throws RemoteException;
  public String [] getStates () throws RemoteException;
  public void reset () throws RemoteException;
  public String [] getOutgoingLinks ()throws RemoteException;
  public String [] getIncomingLinks ()throws RemoteException;
  public void setCPM(Matrix CPM) throws RemoteException;
  public Matrix getCPM() throws RemoteException;
  public Array multiplyTransposeCPM(Array productOfPis)throws RemoteException;
  public Array multiplyCPM(Array productOfPis)throws RemoteException;
  public Array getPI()throws RemoteException;
  public void setPI(Array pi)throws RemoteException;
  public Array getLAMBDA()throws RemoteException;
  public void setLAMBDA(Array lambda)throws RemoteException;
  public Array getBelief() throws RemoteException;
  public void setEvidence(int parentState,int state,boolean learn) throws RemoteException;
  public void setNoEvidence () throws RemoteException;
  public Array getEvidence()throws RemoteException;
}
```

2 Link Component

2.1 LinkEJB

```
////////////////////////////////////////////////////////////////////////
/**
 * Name            LinkEJB
 * Description : The link component administers the PI, LAMBDA and synchronization
 *               for a Bayesian Network Link
 *
 *
 *
 *
 * Change Control
 *
 */

////////////////////////////////////////////////////////////////////////
// Imports

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
////////////////////////////////////////////////////////////////////////
/**
 * Implementation class of the LinkEJB entity bean.
 * Author Anet Potgieter (anet@bayesbean.com)
 * Version 1.0
 *
 * EJB
 *   Type:       Entity
 *   Description: LinkEJB Bean
 *   Home :      LinkHome
 *   Remote:     Link
 *   Persistence: Bean-managed
 *
 */

public class LinkEJB implements EntityBean {

  private String link;
  private String parent;
  private String child;
  private String [] otherParentLinks;
  private String [] otherChildrenLinks;

  private EntityContext context;
  private Connection con;
  private String dbName = "java:comp/env/jdbc/BabeDB";
  private int nrIndexes;
  private int [] cpmIndexes;
```

```
private Array lambda;
private Array pi;
private Array stretchPI;
private String PIFlag = "0";
private String LAMBDAFlag = "0";
private boolean allPIsCalculated = false;
private boolean allLAMBDAsCalculated = false;
private boolean setOutgoingLAMBDAsFlag = false;
private boolean setIncomingPIsFlag = false;
private int nrparentStates;
private int nrchildStates;
private int test = 1;
private boolean trace = false;

////////////////////////////////////////////////////////////////////////
// Business Routines
//

/**********************************************************************
 *  Name : reset
 *  Description : Reset pi, lambda and synchronization
 *                flags for this link
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

public void reset() {

 if (trace) {System.out.println("\n Resetting PI and Lambda for Link" + link);}

 for (int cnt = 0; cnt < pi.length();cnt++){
   pi.set(cnt,1.0) ;}
 for (int cnt = 0; cnt < lambda.length();cnt++){
   lambda.set(cnt,1.0) ;}

 stretchPI();

 PIFlag = "0";
 LAMBDAFlag = "0";
}

/**********************************************************************
 *  Name : setIncomingPIFlags
 *  Description : Set flags for incoming PIS
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

 public void setIncomingPIFlags() {
  setIncomingPIsFlag = true;
 }

 /**********************************************************************
```

```
 *  Name : setOutgoingLAMBDAFlags
 *  Description : Set flags for outgoing LAMBDAS
 *  Input parameters : none
 *  Return : none
 ***********************************************************************/

public void setOutgoingLAMBDAFlags() {
  setOutgoingLAMBDAsFlag = true;
}

/***********************************************************************
 *  Name : allIncomingPIsCalculated
 *  Description : Return flags for incoming PIS
 *  Input parameters : none
 *  Return : boolean
 ***********************************************************************/

public boolean allIncomingPIsCalculated() {
  return allPIsCalculated;
}

/***********************************************************************
 *  Name : allOutgoingLAMBDAsCalculated
 *  Description : Return synchronization flag for outgoing LAMBDAS
 *  Input parameters : none
 *  Return : boolean
 ***********************************************************************/

  public boolean allOutgoingLAMBDAsCalculated() {
    return allLAMBDAsCalculated;
  }

 /***********************************************************************
  *  Name : setPIFlag
  *  Description : Set synchronization flag PI
  *  Input parameters : none
  *  Return : none
  ***********************************************************************/

 public void setPIFlag() {
        PIFlag = "1";
 }

/***********************************************************************
 *  Name : setLAMBDAFlag
 *  Description : Set synchronization flag for LAMBDA
 *  Input parameters : none
 *  Return : none
 ***********************************************************************/

 public void setLAMBDAFlag() {
     LAMBDAFlag = "1";
 }
```

```
/*********************************************************************
 *  Name : getPIFlag
 *  Description : Get synchronization flag for PI
 *  Input parameters : none
 *  Return : String
 *********************************************************************/

 public String getPIFlag() {
   return PIFlag;
 }

/*********************************************************************
 *  Name : getLAMBDAFlag
 *  Description : Get synchronization flag for LAMBDA
 *  Input parameters : none
 *  Return : String
 *********************************************************************/

 public String getLAMBDAFlag () {
   return LAMBDAFlag;
 }

/*********************************************************************
 *  Name : getPI
 *  Description : Get PI for this Link
 *  Input parameters : none
 *  Return : Array
 *********************************************************************/

public Array getPI() {
   return pi;
}

/*********************************************************************
 *  Name : setPI
 *  Description : Set PI for this Link
 *  Input parameters : PI
 *  Return : none
 *********************************************************************/

public void setPI(Array pi) {


  if (pi.length() <= this.pi.length()) {
    for (int cnt = 0; cnt < pi.length();cnt++){
      this.pi.set(cnt,pi.get(cnt));
    }
    if (pi.length() != this.pi.length()) {
      if (trace) {System.out.println("\n LinkEJB : Invalid length for setting PI for Link" + link);}
    }
  }
  stretchPI();
```

```
  if (trace) {
    System.out.println("\n LinkEJB : Setting PI for Link" + link);
    for (int cnt = 0; cnt < pi.length();cnt++){
      System.out.println(pi.get(cnt) + ",");
    }
  }
}

/*****************************************************************
 *  Name : setLAMBDA
 *  Description : Set LAMBDA for this Link
 *  Input parameters : LAMBDA
 *  Return : none
 *****************************************************************/

public void setLAMBDA(Array lambda) {
  this.lambda = lambda;
}

/*****************************************************************
 *  Name : getLAMBDA
 *  Description : get LAMBDA for this Link
 *  Input parameters : none
 *  Return : LAMBDA
 *****************************************************************/

public Array getLAMBDA() {
  return lambda;
}

/*****************************************************************
 *  Name : getParent
 *  Description : get name of this Link's parent
 *  Input parameters : none
 *  Return : String
 *****************************************************************/

 public String getParent() {
  return parent;
 }

/*****************************************************************
 *  Name : getChild
 *  Description : get name of this Link's child
 *  Input parameters : none
 *  Return : String
 *****************************************************************/

 public String getChild() {
      return child;
 }
```

```
/*********************************************************************
 *  Name : getStretchedPi
 *  Description : Return Stretched PI
 *  Input parameters : none
 *  Return : Array
 *********************************************************************/

 public Array getStretchedPi (){
   return this.stretchPI;
 }

/*********************************************************************
 *  Name : getShrinkedLambda
 *  Description : Shrink a LAMBDA
 *  Input parameters : none
 *  Return : Array
 *********************************************************************/

public Array getShrinkedLambda (Array inArr,int nrStates){
  Array shrinkLAMBDA = new Array (nrStates,0.0);
  double sum;

  for (int cnt = 0; cnt < inArr.length(); cnt++){
    int idx = cpmIndexes[cnt];
    sum = shrinkLAMBDA.get(idx) + inArr.get(cnt);
    shrinkLAMBDA.set(idx,sum);
  }
  return shrinkLAMBDA;
}

/*********************************************************************
 *  Name : findOtherParentLinks
 *  Description : Return a list of other outgoing links of this link's
 *                parent node
 *  Input parameters : none
 *  Return : List of Strings
 *********************************************************************/

public String[] findOtherParentLinks(){
  return otherParentLinks;
}

/*********************************************************************
 *  Name : findOtherChildrenLinks
 *  Description : Return a list of other incoming links of this link's
 *                child node
 *  Input parameters : none
 *  Return : List of Strings
 *********************************************************************/

public String[] findOtherChildrenLinks(){
  return otherChildrenLinks;
}
```

```
////////////////////////////////////////////////////////////////////
// The following are required EJB methods, called by the container

/**************************************************************************************
 *  Name : ejbCreate
 *  Description : ejbCreate is called by the EJB container after setEntityContext
 *  Input parameters : Bayesian Network Link Name
 *  Return : String
 **************************************************************************************/

public String ejbCreate(String link)
  throws CreateException {

  try {
    this.link = link;
    selectParentAndChild();
    lambda = new Array(nrparentStates,1.0);
    //lambda = new Array(nrchildStates,1.0);
    pi = new Array(nrparentStates,1.0);
    loadPL("link_pi", "pi", pi);
    loadPL("link_lambda", "lambda", lambda);

    if (trace) {System.out.println("\n EJBCreate LinkEJB PI for Link" + link);
      for (int cnt = 0; cnt < pi.length();cnt++){
        System.out.println(pi.get(cnt) + ",");}}


    getNrIdxs(link);
    getCPMIndexes(link);
    stretchPI();

    loadFlags();
    allPIsCalculated = checkForIncomingPIs();
    allLAMBDAsCalculated = checkForOutgoingLAMBDAs();
    otherParentLinks = getOtherLinks("parent");
    otherChildrenLinks = getOtherLinks("child");

  } catch (Exception ex) {
    throw new EJBException("ejbCreate: " +
    ex.getMessage());
  }
  if (trace) {System.out.println("\nend Link ejbCreate");}
  return link;
}

/*********************************************************************
 *  Name : ejbFindByPrimaryKey
 *  Description : Find the NodeEJB instance that matches the given primary key.
 *  Input parameters : Bayesian Network Node Name
 *  Return : String
 *********************************************************************/
```

```
public String ejbFindByPrimaryKey(String primaryKey)throws FinderException {

  boolean result;

  try {
    result = selectByPrimaryKey(primaryKey);
  } catch (Exception ex) {
    throw new EJBException("ejbFindByPrimaryKey: " +
    ex.getMessage());
  }

  if (result) {
    return primaryKey;
  }
  else {
    throw new ObjectNotFoundException
    ("Row for id " + primaryKey + " not found.");
  }

}

/*****************************************************************
 *  Name : ejbRemove
 *  Description : Called by container before data removed from database.
 *  Input parameters : none
 *  Return : none
 *****************************************************************/

public void ejbRemove() {
  if (trace) {System.out.println("\nLink " + link + "ejbRemove");}
  try {
  }
  catch (Exception ex) {
    throw new EJBException("ejbRemove: " +
    ex.getMessage());
  }
  if (trace) {System.out.println("\nend Link " + link + " ejbRemove");}
}

/*****************************************************************
 *  Name : ejbLoad
 *  Description : Called by container to refresh entity Bean's state.
 *  Input parameters : none
 *  Return : none
 *****************************************************************/

public void ejbLoad() {
  if (trace) {System.out.println("\nLink " + link + " ejbLoad");}
  try {
    //lambda = new Array(nrparentStates,1.0);
    //lambda = new Array(nrchildStates,1.0);
    //pi = new Array(nrparentStates,1.0);
```

```
    loadPL("link_pi", "pi", pi);
    loadPL("link_lambda", "lambda", lambda);
    //getNrIdxs(link);
    //getCPMIndexes(link);
    stretchPI();
    loadFlags();
    allPIsCalculated = checkForIncomingPIs();
    allLAMBDAsCalculated = checkForOutgoingLAMBDAs();

  } catch (Exception ex) {
    throw new EJBException("ejbLoad: " +
    ex.getMessage());
  }
  if (trace) {System.out.println("\nend Link " + link + " ejbLoad");}
}

/*********************************************************************
 *  Name : ejbStore
 *  Description : save Bean's state to database.
 *  Input parameters : none
 *  Return : none
 *********************************************************************/

public void ejbStore() {
  if (trace) {System.out.println("\nLink " + link + " ejbStore");}
  try {
    storePL("link_pi", "pi", pi);
    storePL("link_lambda", "lambda", lambda);
    setFlags(this.PIFlag, this.LAMBDAFlag);
    if (setIncomingPIsFlag  == true)
    {
      setPL("pi_flag", "child",this.child,"0");
      setIncomingPIsFlag  = false;
    }

    if (setOutgoingLAMBDAsFlag == true)
    {
      setPL("lambda_flag", "parent",this.parent,"0");
      setOutgoingLAMBDAsFlag = false;
    }
  }
  catch (Exception ex) {
    throw new EJBException("ejbLoad: " +
    ex.getMessage());
  }
  if (trace) {System.out.println("\nend Link " + link + " ejbStore");}
}

/*********************************************************************
 *  Name : ejbPostCreate
 *  Description : Called by container after ejbCreate.
 *  Input parameters : none
 *  Return : none
```

```
   ***********************************************************************/

   public void ejbPostCreate(String link) { }

////////////////////////////////////////////////////////////////////////
// The following are callback methods, called by the container to
// notify the Bean that some event is about to occur.

  /**********************************************************************
   *  Name : setEntityContext
   *  Description : Called by container to set Bean context.
   *  Input parameters : context
   *  Return : none
   **********************************************************************/

  public void setEntityContext(EntityContext context) {
    if (trace) {System.out.println("\nLink " + link + " setEntityContext");}
    this.context = context;
    try {
      makeConnection();
    }
    catch (Exception ex) {
      throw new EJBException("Unable to connect to database. " +
      ex.getMessage());
    }
    if (trace) {System.out.println("\nend Link " + link + " setEntityContext");}
  }

  /**********************************************************************
   *  Name : unsetEntityContext
   *  Description : Called by container to unset Bean context.
   *  Input parameters : context
   *  Return : none
   **********************************************************************/

  public void unsetEntityContext() {
    if (trace) {System.out.println("\nLink " + link + " unsetEntityContext");}
    try {
      con.close();
    }
    catch (SQLException ex) {
      throw new EJBException("unsetEntityContext: " + ex.getMessage());
    }
    if (trace) {System.out.println("\nend Link " + link + " unsetEntityContext");}
  }

  /**********************************************************************
   *  Name : ejbActivate
   *  Description : Called by container before Bean swapped into memory.
   *  Input parameters : none
   *  Return : none
   **********************************************************************/
```

```
  public void ejbActivate() {
    if (trace) {System.out.println("\nLink " + link + " ejbActivate");}
    link = (String)context.getPrimaryKey();
    if (trace) {System.out.println("\nend Link ejbActivate");}
  }

  /*********************************************************************
   *  Name : ejbPassivate
   *  Description : Called by container before Bean swapped into storage.
   *  Input parameters : none
   *  Return : none
   *********************************************************************/

  public void ejbPassivate() {
    if (trace) {System.out.println("\nLink " + link + " ejbPassivate");}
    link = null;
    if (trace) {System.out.println("\nend Link " + link + " ejbPassivate");}
  }

////////////////////////////////////////////////////////////////////////
/********************** Database Routines ************************/

  /*********************************************************************
   *  Name : makeConnection
   *  Description : Connect to the database
   *  Input parameters : none
   *  Return : none
   *********************************************************************/

  private void makeConnection() throws NamingException, SQLException {
    InitialContext ic = new InitialContext();
    DataSource ds = (DataSource) ic.lookup(dbName);
    con =  ds.getConnection();
  }

  /*********************************************************************
   *  Name : getNrIdxs
   *  Description : Determine Number of Indexes for this link
   *  Input parameters : link
   *  Return : none
   *********************************************************************/

  private void getNrIdxs(String link) throws SQLException {
    if (trace) {System.out.println("\nLink " + link + "getNrIdxs");}
    //Determine nr indexes for this link
    String selectStatement = "select count(nr) from link_indexes where link = ?";
    PreparedStatement prepStmt =
    con.prepareStatement(selectStatement);
    prepStmt.setString(1, link);

    ResultSet rs = prepStmt.executeQuery();

    if (rs.next()) {
```

```
    this.nrIndexes = rs.getInt(1);
    if (trace) {System.out.println("\nLink " + link + " Nr of Indexes = " + nrIndexes + "\n");}
   }
   prepStmt.close();
}

/**********************************************************************
 *  Name : getCPMIndexes
 *  Description : Retrieve CPM Indexes
 *  Input parameters : link
 *  Return : none
 **********************************************************************/

private void getCPMIndexes(String link) throws SQLException {
  if (trace) {System.out.println("\nLink " + link + "getCPMIndexes");}
  cpmIndexes = new int[nrIndexes];

  String selectStatement = "select nr, index " +
    "from link_indexes where link = ? order by nr";
  PreparedStatement prepStmt = con.prepareStatement(selectStatement);

  prepStmt.setString(1, link);
  ResultSet rs = prepStmt.executeQuery();

  for (int i=0; i < nrIndexes; i++){
    rs.next();
    int idx = rs.getInt(2);
    cpmIndexes[i] = idx;
  }

  prepStmt.close();

  if (trace) {System.out.println("\nend Link getCPMIndexes");}
}

/**********************************************************************
 *  Name : selectByPrimaryKey
 *  Description : Select Link by Primary Key
 *  Input parameters : primaryKey
 *  Return : boolean
 **********************************************************************/

private boolean selectByPrimaryKey(String primaryKey)
  throws SQLException {
  String selectStatement =
    "select link " +
    "from links where link = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, primaryKey);

  ResultSet rs = prepStmt.executeQuery();
  boolean result = rs.next();
```

```
  prepStmt.close();
  return result;
}

/*********************************************************************
 *  Name : insertPL
 *  Description : Insert PI/LAMBDA for this link
 *  Input parameters : Table Name, PI/LAMBDA
 *  Return : none
 *********************************************************************/

private void insertPL (String TableName,Array PL) throws SQLException {
  String selectStatement =
    "select link " +
    "from " + TableName + " where link = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, link);

  ResultSet rs = prepStmt.executeQuery();

  if (rs.next()) {
    prepStmt.close();
    deletePL(TableName);
  }
  else {
    prepStmt.close();
  }

  String insertStatement = "insert into " + TableName + " values ( ? , ? , ? )";
  prepStmt = con.prepareStatement(insertStatement);
  prepStmt.setString(1, link);
  for (int cnt = 1;cnt <= PL.length();cnt++) {
    prepStmt.setInt(2, cnt);
    prepStmt.setDouble(3, PL.get(cnt-1));
    prepStmt.executeUpdate();
  }
  prepStmt.close();
}

/*********************************************************************
 *  Name : deletePL
 *  Description : Delete PI/LAMBDA for this link
 *  Input parameters : Table Name
 *  Return : none
 *********************************************************************/

private void deletePL(String TableName) throws SQLException {
  String deleteStatement =
    "delete from " + TableName + " where link = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(deleteStatement);
```

```
  prepStmt.setString(1, this.link);
  prepStmt.executeUpdate();
  prepStmt.close();
}

/*********************************************************************
 *  Name : loadPL
 *  Description : Load PI or LAMBDA for this link
 *  Input parameters : Table Name, Field Name, PI/LAMBDA
 *  Return : none
 *********************************************************************/

private void loadPL(String TableName, String Field, Array arr) throws SQLException {
  String selectStatement =
    "select link, idx, " + Field +
    " from " + TableName + " where link = ? order by link,idx";

  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);

  prepStmt.setString(1, this.link);

  ResultSet rs = prepStmt.executeQuery();
  int cnt = 1;
  while (rs.next()) {
    if (cnt <= arr.length()) {
      arr.set(cnt - 1, rs.getDouble(3));
      cnt++;}
  }
  prepStmt.close();
}

/*********************************************************************
 *  Name : storePL
 *  Description : Store PI or LAMBDA for this link
 *  Input parameters : Table Name, Field Name, PI/LAMBDA
 *  Return : none
 *********************************************************************/

private void storePL(String TableName, String Field, Array arr) throws SQLException {
  int rowCount = 0;

  String updateStatement = "update " + TableName + " set " + Field + " =  ? " +
    "where link = ? and idx = ?";

  PreparedStatement prepStmt =
  con.prepareStatement(updateStatement);
  prepStmt.setString(2, this.link);
  for (int cnt = 1;cnt <= arr.length(); cnt++) {
    prepStmt.setInt(3, cnt);
    prepStmt.setDouble(1, arr.get(cnt - 1));
    rowCount = prepStmt.executeUpdate();
  }
```

```
  prepStmt.close();
}

/*********************************************************************
 *  Name : setPL
 *  Description : Update pi-flags/lambda_flag for link
 *  Input parameters : Field Name, Relation specification,relation,val
 *  Return : none
 *********************************************************************/

private void setPL(String Field, String relationSpec,String relation,String val) throws SQLException {
  int rowCount;

  String updateStatement = "update links set " + Field + " =  ? " +
    "where " + relationSpec + " = ?";

  PreparedStatement prepStmt =
  con.prepareStatement(updateStatement);
  prepStmt.setString(1, val);
  prepStmt.setString(2, relation);

  rowCount = prepStmt.executeUpdate();
  prepStmt.close();

  if (rowCount == 0) {
    throw new EJBException("Resetting incoming pi or lambda flags for child " + child + " failed.");
  }

}

/*********************************************************************
 *  Name : selectParentAndChild
 *  Description : Select the parent and child for this link
 *  Input parameters : none
 *  Return : none
 *********************************************************************/


private void selectParentAndChild()
  throws SQLException {

  String selectStatement =
    "select parent,child " +
    "from links where link = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, link);

  ResultSet rs = prepStmt.executeQuery();
  rs.next();
  parent = rs.getString(1);
  child = rs.getString(2);
  prepStmt.close();
```

```
    selectStatement =
      "select count(state)" +
      "from states where node = ? ";
    prepStmt =
    con.prepareStatement(selectStatement);
    prepStmt.setString(1, parent);

    rs = prepStmt.executeQuery();
    rs.next();
    nrparentStates = rs.getInt(1);
    if (trace) {System.out.println("\nLink " + link + "'s Parent " + parent + " has " + nrparentStates + "
States");}
    prepStmt.setString(1, child);
    rs = prepStmt.executeQuery();
    rs.next();
    nrchildStates = rs.getInt(1);
    prepStmt.close();
    if (trace) {System.out.println("\nLink " + link + "'s Child has " + nrchildStates + " States");}
  }

  /*****************************************************************************
   *  Name : getOtherLinks
   *  Description : Get other outgoing links of parent / other incoming links of child
   *  Input parameters : relation
   *  Return : List of Strings
   *****************************************************************************/


  private String[] getOtherLinks(String relation){
  String [] otherLinks;

  try {
    otherLinks = selectOtherLinks(relation);
  }
  catch (Exception ex) {
    throw new EJBException("getOtherLinks: " +
    ex.getMessage());
  }
  return otherLinks;
  }

  /*****************************************************************************
   *  Name : selectOtherLinks
   *  Description : Finds other sibling links for this link
   *  Input parameters : relation
   *  Return : List of Strings
   *****************************************************************************/

  private String[] selectOtherLinks(String relation) throws SQLException {
    // Determine nr other incoming/outgoing links for this link
    String [] a = new String[0];
    int nrOtherLinks = 0;
```

```
  String selectStatement = "select count(link) from links where " + relation + " = ? and not link = ?";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);

  if (relation.compareTo("parent") == 0) {
    prepStmt.setString(1, parent);}
  else {
    prepStmt.setString(1, child);}

  prepStmt.setString(2, link);
  ResultSet rs = prepStmt.executeQuery();

  if (rs.next()) {
    nrOtherLinks = rs.getInt(1);
  }
  prepStmt.close();

  // Retrieve the siblings and return in an array

  if (nrOtherLinks == 0) {
    return a;
  }

  selectStatement = "select link from links where " + relation + " = ? and not link = ?";
  prepStmt = con.prepareStatement(selectStatement);

  if (relation.compareTo("parent") == 0) {
    prepStmt.setString(1, parent);}
  else {
    prepStmt.setString(1, child);}

  prepStmt.setString(2, link);
  rs = prepStmt.executeQuery();
  a = new String[nrOtherLinks];
  int cnt = 0;

  while (rs.next()) {
    String otherLink = rs.getString(1);
    a[cnt] = otherLink;
    cnt++;
  }
  prepStmt.close();
  return a;
}

/*****************************************************************************
 *  Name : selectOtherLinks
 *  Description : Store PI and LAMBDA flags for this link
 *  Input parameters : PIFlag, LambdaFlag
 *  Return : none
 *****************************************************************************/

private void setFlags(String PIFlg, String LAMBDAFlg) throws SQLException {
```

```
  int rowCount = 0;

  String updateStatement = "update links set pi_flag = ?, lambda_flag = ?" +
    " where link = ?";
  PreparedStatement prepStmt =
  con.prepareStatement(updateStatement);
  prepStmt.setString(1, PIFlg);
  prepStmt.setString(2, LAMBDAFlg);
  prepStmt.setString(3, this.link);
  rowCount = prepStmt.executeUpdate();
  prepStmt.close();

  if (rowCount == 0) {
    throw new EJBException("Storing pi or lambda for link " + this.link + " failed.");
  }
}

/*****************************************************************************
 *  Name : loadFlags
 *  Description : load Flags for this link
 *  Input parameters : none
 *  Return : none
 *****************************************************************************/

private void loadFlags() throws SQLException {

  String selectStatement =
    "select pi_flag, lambda_flag " +
    "from links where link = ? ";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, link);

  ResultSet rs = prepStmt.executeQuery();
  rs.next();
  this.PIFlag = rs.getString(1);
  this.LAMBDAFlag = rs.getString(2);
  prepStmt.close();

}

/*****************************************************************************
 *  Name : checkForIncomingPIs
 *  Description : See if all incoming PI's were calculated for child
 *  Input parameters : none
 *  Return : boolean
 *****************************************************************************/

private boolean checkForIncomingPIs() throws SQLException {

  boolean returnVal;
  String selectStatement =
```

```
  "select link " +
  "from links where not parent = ? and child = ? and pi_flag = ?";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, parent);
  prepStmt.setString(2, child);
  prepStmt.setString(3,"0");

  ResultSet rs = prepStmt.executeQuery();

  if (rs.next()){
    returnVal = false; }
  else {
    returnVal = true ; }
  prepStmt.close();

  return returnVal;
}

/*****************************************************************************
 *  Name : checkForOutgoingLAMBDAs
 *  Description : See if all outgoing LAMBDA's were calculated for parent
 *  Input parameters : none
 *  Return : boolean
 *****************************************************************************/

private boolean checkForOutgoingLAMBDAs() throws SQLException {
  boolean returnVal;

  String selectStatement =
    "select link " +
    "from links where not child = ? and parent = ? and lambda_flag = ?";
  PreparedStatement prepStmt =
  con.prepareStatement(selectStatement);
  prepStmt.setString(1, child);
  prepStmt.setString(2, parent);
  prepStmt.setString(3,"0");

  ResultSet rs = prepStmt.executeQuery();

  if (rs.next()){
    returnVal = false; }
  else {
    returnVal = true ; }
  prepStmt.close();

  return returnVal;
}

/*****************************************************************************
 *  Name : stretchPI
 *  Description : Stretch PI for this parent so that it can by multiplied with PI's
 *                of siblings
```

```
   *  Input parameters : none
   *  Return : none
   ********************************************************************************/

  private void stretchPI() {
    if (trace) {System.out.println("\n LinkEJB stretching PI Link " + link + " NrIndexes: " + nrIndexes);}
    stretchPI = new Array (nrIndexes,1.0);

    if (trace) {System.out.println("\n LinkEJB: PI for Link " + link );

    for (int cnt = 0; cnt < pi.length();cnt++){
      System.out.println(pi.get(cnt) + ",");}

    System.out.println("\n LinkEJB: END PI for Link " + link );}

    for (int cnt = 0; cnt < nrIndexes; cnt++){
      int idx = cpmIndexes[cnt];

      if (idx < pi.length ())
        {stretchPI.set(cnt,pi.get(idx));}
      else
        {stretchPI.set(cnt,1.0);
      System.out.println("\n LinkEJB: WARNING  setting stretched PI[" + cnt + "] to 1.0 for Link " + link
);}
    }
    if (trace) {System.out.println("\nend LinkEJB stretchedPI");}
  }
} // LinkEJB
```

2.2 Link Home

```
////////////////////////////////////////////////////////////////////////
/**
 * Name         LinkHome
 * Description : HOME INTERFACE FOR THE LINK ENTITY BEAN CLASS
 *
 *
 *
 *
 * Change Control
 *
 */

////////////////////////////////////////////////////////////////////////

import java.rmi.RemoteException;
import javax.ejb.*;
import java.util.Collection;

public interface LinkHome extends EJBHome {

public Link create(String link)
      throws CreateException, RemoteException;

public Link findByPrimaryKey(String link)
      throws FinderException, RemoteException;


}
```

2.3 Link Remote

```
////////////////////////////////////////////////////////////////////////
/**
 * Name         Link
 * Description : REMOTE INTERFACE FOR THE LINK ENTITY BEAN CLASS
 *
 *
 *
 *
 * Change Control
 *
 */

////////////////////////////////////////////////////////////////////////

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

  public interface Link extends EJBObject {
    public void reset () throws RemoteException;
    public String[] findOtherParentLinks()throws RemoteException;
    public String[] findOtherChildrenLinks()throws RemoteException;
    public Array getPI()throws RemoteException;
    public void setPI(Array pi) throws RemoteException;
    public void setLAMBDA(Array lambda) throws RemoteException;
    public Array getLAMBDA()throws RemoteException;
    public String getParent()throws RemoteException;
    public String getChild()throws RemoteException;
    public void setPIFlag() throws RemoteException;
    public void setLAMBDAFlag() throws RemoteException;
    public String getPIFlag() throws RemoteException;
    public String getLAMBDAFlag () throws RemoteException;
    public Array getStretchedPi ()throws RemoteException;
    public Array getShrinkedLambda (Array inArr,int nrStates) throws RemoteException;
    public void setIncomingPIFlags () throws RemoteException;
    public void setOutgoingLAMBDAFlags()throws RemoteException;
    public boolean allIncomingPIsCalculated() throws RemoteException;
    public boolean allOutgoingLAMBDAsCalculated() throws RemoteException;
}
```

3 Belief Propagation Component

```
////////////////////////////////////////////////////////////////////////////
/**
 * Name            MessageBean
 * Description : This belief propagation component implements the local belief
 *               propagation on a link between a parent and a child node
 *
 *
 *
 *
 * Change Control
 *
 */

////////////////////////////////////////////////////////////////////
// Imports


import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.ejb.CreateException;
import javax.naming.*;
import javax.jms.*;
import java.util.*;
import javax.rmi.PortableRemoteObject;
import javax.transaction.*;

////////////////////////////////////////////////////////////////////
/**
 * The MessageBean class is a message-driven bean.  It implements
 * the javax.ejb.MessageDrivenBean and javax.jms.MessageListener
 * interfaces. It is defined as public (but not final or
 * abstract).  It defines a constructor and the methods ejbCreate,
 * onMessage, setMessageDrivenContext, and ejbRemove.
 */

public class MessageBean implements MessageDrivenBean, MessageListener {
  QueueConnection     queueConnection = null;
  private Context newContext;

  private transient MessageDrivenContext mdc = null;
  private boolean trace = false;
  private boolean trace2 = true;
  private boolean propagateFlag = false;

  /**
  * Constructor, which is public and takes no arguments.
  */
```

```
public MessageBean() {
//System.out.println("In MessageBean.MessageBean()");
}

/**********************************************************************
 *  Name : setMessageDrivenContext method
 *  Description : declared as public (but not final or static)
 *  Input parameters : javax.ejb.MessageDrivenContext
 *  Return : none
 **********************************************************************/

public void setMessageDrivenContext(MessageDrivenContext mdc)
{
  this.mdc = mdc;
}

/**********************************************************************
 *  Name : ejbCreate method
 *  Description : declared as public (but not final or static)
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

public void ejbCreate() {

  try {
    newContext = new InitialContext();
    if (trace) {System.out.println("In MessageBean.ejbCreate()");}

    }
    catch (Throwable t) {

      // JMSException or NamingException could be thrown
      t.printStackTrace();
  }
}

/**********************************************************************
 *  Name : ejbCreate method
 *  Description : Closes the QueueConnection.
 *  Input parameters : none
 *  Return : none
 **********************************************************************/

public void ejbRemove() {
  Link link;

  try {

  if (trace) {System.out.println("In MessageBean.ejbRemove()");}

  if(queueConnection != null) {
    queueConnection.close();};
```

```
  }
  catch (Throwable t) {

    // JMSException or NamingException could be thrown
    t.printStackTrace();
  }

}

/*********************************************************************
 *  Name : onMessage method
 *  Description : Handles the incoming message
 *  Input parameters : inMessage the incoming message
 *  Return : none
 *********************************************************************/

public void onMessage(Message inMessage) {
  TextMessage msg = null;
  Queue queue = null;

  try {

    queue = (Queue)inMessage.getJMSDestination();
    String dest = queue.getQueueName();

    if (inMessage instanceof TextMessage) {
      msg = (TextMessage) inMessage;

      String tag = msg.getText();
      if (trace2) {System.out.println("MessageBean : " + dest + " received a Message" + tag);}

      if (tag.compareTo("PI") == 0) {
        handlePI(dest);
      }

      if (tag.compareTo("LAMBDA") == 0) {
        handleLAMBDA(dest);
      }

      if (tag.compareTo("PAUSE") == 0)
      {
       propagateFlag = false;
      }

      if (tag.compareTo("RESUME") == 0)
      {
       propagateFlag = true;
      }
    }
  }
  catch (Throwable t) {
```

```
    // JMSException could be thrown
    t.printStackTrace();

  }
}

/**********************************************************************
 *  Name : handlePI
 *  Description : Handles the incoming PI message
 *  Input parameters : Queue name
 *  Return : none
 **********************************************************************/

private void handlePI(String destination)
{

  int cnt = 0;
  Link link;
  Link otherLink;
  Array LinkPI = new Array(0);
  Array productOfLambdas = new Array(0);
  Array productOfPis = new Array(0);
  Array piX = new Array(0);
  String parent;
  String child;

  Object linkObjref;
  Object nodeObjref;
  NodeHome nodeHome;
  LinkHome linkHome;
  Node node;
  Node childNode;
  Node parentNode;
  String [] otherIncomingLinks = new String[0];
  String [] otherOutgoingLinks = new String[0];
  String [] ChildOutgoingLinkNames = new String[0];
  String [] ChildIncomingLinkNames = new String[0];

  try {

    linkObjref = newContext.lookup("java:comp/env/ejb/LinkBean");
    linkHome = (LinkHome)PortableRemoteObject.narrow(linkObjref, LinkHome.class);

    nodeObjref = newContext.lookup("java:comp/env/ejb/NodeBean");
    nodeHome = (NodeHome)PortableRemoteObject.narrow(nodeObjref, NodeHome.class);

    link = linkHome.create(destination);
    parent = link.getParent();
    child = link.getChild();

    // Get a list of other outgoing links
    // ----------------------------------
    otherOutgoingLinks = link.findOtherParentLinks();
```

```
// Get a list of other incoming links
// ----------------------------------
otherIncomingLinks = link.findOtherChildrenLinks();
link.remove();

// calculate product of lambdas for other outgoing links
// -----------------------------------------------------

for (cnt = 0; cnt <  otherOutgoingLinks.length; cnt++) {

  otherLink = linkHome.create(otherOutgoingLinks[cnt]);
  if (cnt == 0) {
    productOfLambdas = otherLink.getLAMBDA();
  }
  else {
    productOfLambdas = productOfLambdas.termProduct(otherLink.getLAMBDA());
  }
  otherLink.remove();
}

if (trace) {System.out.println(destination + "Product of Lambdas: [");
  for (cnt = 0; cnt < productOfLambdas.length();cnt++){
    System.out.println(productOfLambdas.get(cnt) + ",");}}


// Get parent's PI
// -----------------------------------

parentNode = nodeHome.create(parent);
//piX = parentNode.getPI();
piX = parentNode.getBelief();

String [] parentIncomingLinkNames = parentNode.getIncomingLinks ();

// calculate PI for this link
// --------------------------

if (otherOutgoingLinks.length > 0 ) {
  LinkPI = piX.termProduct(productOfLambdas);}
else {
  LinkPI = piX;
}

link = linkHome.create(destination);

// Set PI for this link
// --------------------
if (trace2) {System.out.println(destination + ": PI : [");
  for (cnt = 0; cnt < LinkPI.length();cnt++){
    System.out.println(LinkPI.get(cnt) + ",");}}

link.setPI(LinkPI);
```

```
// Set PI flag for this link
// ------------------------
link.setPIFlag();

// Test if other PI's were calculated for child's other incoming links
// -------------------------------------------------------------------

boolean testLinks = link.allIncomingPIsCalculated();

if (testLinks) {
  link.setIncomingPIFlags();
}

if (testLinks) {
  // Calculate PI for child node
  // ---------------------------


  if (trace2) {System.out.println("Can now calculate child " + child + "'s PI !");}
  productOfPis = link.getStretchedPi();
  link.remove();

  for (cnt = 0; cnt <  otherIncomingLinks.length; cnt++) {

    // For each other incoming link
    // ----------------------------
    otherLink = linkHome.create(otherIncomingLinks[cnt]);

    // calculate product of pis of other incoming links
    // ------------------------------------------------

    productOfPis = productOfPis.termProduct(otherLink.getStretchedPi());
    otherLink.remove();

  }

  childNode = nodeHome.create(child);
  ChildOutgoingLinkNames = childNode.getOutgoingLinks ();
  ChildIncomingLinkNames = parentNode.getIncomingLinks ();

  Array pi = childNode.multiplyTransposeCPM(productOfPis);

  if (trace2) {System.out.println(child + "'s PI : [");
    for (cnt = 0; cnt < pi.length();cnt++){
      System.out.println(pi.get(cnt) + ",");}}

  childNode.setPI(pi);

  // PROPAGATE TO CHILDREN
  // ---------------------

  // If not a leaf node, propagate PIs downwards
```

```
      if (ChildOutgoingLinkNames.length > 0 ) {
        for (int childCnt = 0; childCnt < ChildOutgoingLinkNames.length; childCnt++) {
          System.out.println("\n Propagating PI message to " + ChildOutgoingLinkNames[childCnt] +
          " from " + destination);
          sendMsg(ChildOutgoingLinkNames[childCnt],"PI",destination);
        }
      }
      else
      {
        if (propagateFlag) {
          for (int childCnt = 0; childCnt < ChildIncomingLinkNames.length; childCnt++) {
            System.out.println("\n Propagating LAMBDA message to " +
ChildIncomingLinkNames[childCnt] +
            " from " + destination);
            sendMsg(ChildIncomingLinkNames[childCnt],"LAMBDA",destination);
          }
        }
      }
    }
    else {
      link.remove();
      System.out.println(destination + ": Can not calculate " + child + "'s PI yet !");
    }

  } catch (Throwable t) {
    // JMSException could be thrown
    t.printStackTrace();
  }
 }

 /**********************************************************************
  *  Name : handleLAMBDA
  *  Description : Handles the incoming LAMBDA message
  *  Input parameters : Queue name
  *  Return : none
  **********************************************************************/

 private void handleLAMBDA(String destination)
 {
  int cnt = 0;
  Link link;
  Link otherLink;
  Array LinkLambda = new Array(0);
  Array productOfLambdas = new Array(0);
  Array productOfPis = new Array(0);
  Array lambdaY = new Array(0);
  Array longLambda = new Array(0);
  String parent;
  String child;

  Object linkObjref;
  Object nodeObjref;
```

```
NodeHome nodeHome;
LinkHome linkHome;
Node node;
Node childNode;
Node parentNode;
String [] otherIncomingLinks = new String[0];
String [] otherOutgoingLinks = new String[0];
String [] parentIncomingLinkNames = new String[0];
String [] parentOutgoingLinkNames = new String[0];

try {


  linkObjref = newContext.lookup("java:comp/env/ejb/LinkBean");
  linkHome = (LinkHome)PortableRemoteObject.narrow(linkObjref, LinkHome.class);

  nodeObjref = newContext.lookup("java:comp/env/ejb/NodeBean");
  nodeHome = (NodeHome)PortableRemoteObject.narrow(nodeObjref, NodeHome.class);


  link = linkHome.create(destination);

  parent = link.getParent();
  child = link.getChild();

  // Get a list of other outgoing links
  // ----------------------------------
  otherOutgoingLinks = link.findOtherParentLinks();

  // Get a list of other incoming links
  // ----------------------------------
  otherIncomingLinks = link.findOtherChildrenLinks();

  // calculate product of pis for other incoming links
  // -------------------------------------------------
  productOfPis = link.getStretchedPi();
  link.remove();

  for (cnt = 0; cnt <  otherIncomingLinks.length; cnt++) {

    // For each other incoming link
    // ----------------------------
    otherLink = linkHome.create(otherIncomingLinks[cnt]);

    // calculate product of pis of other incoming links
    // ------------------------------------------------


    if (cnt == 0) {
      productOfPis = otherLink.getStretchedPi();
    }
    else
    {
```

```
   productOfPis = productOfPis.termProduct(otherLink.getStretchedPi());
  }
  otherLink.remove();

}

if ((trace) & (otherIncomingLinks.length > 0)) {System.out.println(destination + ": productOfPis : [");
  for (cnt = 0; cnt < productOfPis.length();cnt++){
    System.out.println(productOfPis.get(cnt) + ",");}}

childNode = nodeHome.create(child);


// calculate LAMBDA for this link
// ------------------------------

lambdaY = childNode.getLAMBDA();

if (trace) {System.out.println(destination + "Got Child " + child + "'s LAMBDA : [");
  for (cnt = 0; cnt < lambdaY.length();cnt++){
  System.out.println(" Child " + child + lambdaY.get(cnt) + ",");}}


String [] childOutgoingLinkNames = childNode.getOutgoingLinks ();

Array lambdaProd = childNode.multiplyCPM(lambdaY);

if (otherIncomingLinks.length > 0 ) {
  longLambda = lambdaProd.termProduct(productOfPis);}
else {
  longLambda = lambdaProd; }

parentNode = nodeHome.create(parent);
int nrStates = parentNode.getNrStates ();
parentNode.remove();

link = linkHome.create(destination);
LinkLambda = link.getShrinkedLambda (longLambda,nrStates);

// Set LAMBDA for this link
// --------------------
if (trace2) {System.out.println(destination + ": LAMBDA : [");
  for (cnt = 0; cnt < LinkLambda.length();cnt++){
    System.out.println(LinkLambda.get(cnt) + ",");}}

link.setLAMBDA(LinkLambda);

// Set LAMBDA flag for this link
// -------------------------

link.setLAMBDAFlag();

// Test if other LAMBDA's were calculated for parent's other outgoing links
```

```
// ------------------------------------------------------------------

boolean testLinks = link.allOutgoingLAMBDAsCalculated();
if (testLinks) {
  link.setOutgoingLAMBDAFlags();
}

if (testLinks) {
  // Calculate LAMBDA for parent node
  // --------------------------------
  System.out.println(destination + ":Can now calculate " + parent + "'s LAMBDA !");
  productOfLambdas = link.getLAMBDA();
  link.remove();


  for (cnt = 0; cnt <  otherOutgoingLinks.length; cnt++) {

    // For each other outgoing link
    // ----------------------------
    otherLink = linkHome.create(otherOutgoingLinks[cnt]);

    // calculate product of lambdas of other outgoing links
    // ----------------------------------------------------

    productOfLambdas = productOfLambdas.termProduct(otherLink.getLAMBDA());
    otherLink.remove();

  }

  parentNode = nodeHome.create(parent);
  parentIncomingLinkNames = parentNode.getIncomingLinks ();
  parentOutgoingLinkNames = parentNode.getOutgoingLinks ();

  Array lambda = productOfLambdas;

  if (trace2) {System.out.println(parent + ": LAMBDA : [");
    for (cnt = 0; cnt < lambda.length();cnt++){
      System.out.println(lambda.get(cnt) + ",");}
  }

  parentNode.setLAMBDA(lambda);

  // If not a root node, propagate lambdas upwards

  if (parentIncomingLinkNames.length > 0 ) {

    for (int parentCnt = 0; parentCnt < parentIncomingLinkNames.length; parentCnt++) {

    if (trace2) {System.out.println("\n Propagating LAMBDA message to " +
parentIncomingLinkNames[parentCnt] + " from " + destination);}
      sendMsg(parentIncomingLinkNames[parentCnt],"LAMBDA",destination);

    }
```

```
      }
      else { // root node - propagate PI's downwards
        for (int childCnt = 0; childCnt < parentOutgoingLinkNames.length; childCnt++) {

        if (trace2) {System.out.println("\n Propagating PI message to " +
parentOutgoingLinkNames[childCnt] + " from " + destination);}
          sendMsg(parentOutgoingLinkNames[childCnt],"PI",destination);
        }
      }
    }
    else {
      link.remove();
      System.out.println(destination + ":Can not calculate " + parent + "'s LAMBDA yet !");}
  }
  catch (Throwable t) {

    // JMSException could be thrown
    t.printStackTrace();

  }

 }
 /**********************************************************************
  *  Name : sendMsg
  *  Description : Send Message to queue
  *  Input parameters : Link Name, Message,
  *  Return : none
  **********************************************************************/


 private void sendMsg(String linkName,String msg, String dest) {
  Context               jndiContext = null;
  QueueConnectionFactory  queueConnectionFactory = null;
  QueueConnection         queueConnection = null;
  QueueSession            queueSession = null;
  Queue                 queue = null;
  QueueSender             queueSender = null;
  TextMessage             message = null;
  String queueName = "java:comp/env/jms/" + linkName;
  String queueConnectionFactoryName = "java:comp/env/jms/" + linkName + "CF";
  /*
  * Create a JNDI InitialContext object if none exists yet.
  */
  jndiContext = newContext;
  /*
  * Look up connection factory and queue.  If either does
  * not exist, exit.
  */
  try {
    queue = (Queue)jndiContext.lookup(queueName);
    queueConnectionFactory =
(QueueConnectionFactory)jndiContext.lookup(queueConnectionFactoryName);
  }
```

```
    catch (NamingException e) {
      System.out.println("Could not create JNDI " + "context: " + e.toString() + "from " + dest);
      System.exit(1);
    }

    try {
      queueConnection = queueConnectionFactory.createQueueConnection();
      queueSession =
      queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
      queueSender = queueSession.createSender(queue);

      message = queueSession.createTextMessage();
      message.setJMSDestination(queue);
      message.setText(msg);
      queueSender.send(message);
      if (queueSession != null) {
        queueSession.close();
      }
      if (queueConnection != null) {
        queueConnection.close();
      }
    }
    catch (JMSException e)
    {

      System.out.println("Exception occurred: " +
      e.toString());

    }
  }

}
```