Changes to MASS for exchangeBoundary()

Constants.java

ADD:

```
public static final int EXCHANGE_BOUNDARY = 16;
```

Message.java

ADD:

```
// Exchange Boundary Variables
private Vector<int[]>       EB_DESTINATIONS = null;
private int                 BNDRYLENGTH;
private boolean             WRAPEDGES;


// Create ExchangeBoundary Message
public void createExchangeBoundaryMessage( int functionId,  Vector<int[]> destinations )
{
        ACTION                  = Constants.EXCHANGE_BOUNDARY;
        FUNCTION_ID             = functionId;
        EB_DESTINATIONS         = destinations;
}

public ArrayList<RemoteExchangeRequest> getExchangeBoundaryMessage()    {
        return (ArrayList<RemoteExchangeRequest>) this.message.get(Constants.EXCHANGE_BOUNDARY);
}

public Vector<int[]> getEBDestinations() { return EB_DESTINATIONS; }
public int getBndryLength() { return BNDRYLENGTH; }
public boolean wrapEdges() { return WRAPEDGES; }
```

UPDATE:

```
public void createInitializationMessage(int[] size,  String arrayType,  String placeType,  int handle,
 String className,  Object argument,  HashMap<Integer, String> networkMap,  HashMap<String, Integer> nodePidMap,
int dlbCnt,  boolean historyBased,  boolean windowBased,  boolean slopeBased) {
        ...
        NETWORK_MAP = networkMap;
        ...
}
```

to

```
 public void createInitializationMessage(int[] size,  String arrayType,  String placeType,  int handle,
String className,  Object argument,  HashMap<String, Integer> nodePidMap,  int bndryLength,  boolean wrap,
int dlbCnt,  boolean historyBased,  boolean windowBased,  boolean slopeBased ) {
        ...
        BNDRYLENGTH = bndryLength;
        WRAPEDGES = wrap;
}       ...
```

REMOVE:

```
private HashMap<Integer, String> NETWORK_MAP = null;       // map of user array index, host name responsible
public HashMap<Integer, String> getNetworkMap() { return this.NETWORK_MAP; }
```

MProcess.java

ADD:

```
case  Constants.EXCHANGE_BOUNDARY:
        //log("=============Calling ExchangeBoundary: FuncID: " + m.getFunctionId()+ "=============");
        PLACES.exchangeBoundary(1, m.getFunctionId(), m.getEBDestinations());
        //log("=============Finished ExchangeBoundary: FuncID: " + m.getFunctionId() + "=============");
        break;
```

UPDATE:

```
PLACES = new Places(m.getHandle(), m.getClassName(), m.getArgument(), m.getSize());
```

to

```
if( m.getBndryLength() > 0 ) {
        PLACES = new Places( m.getHandle(), m.getClassName(), m.getArgument(), m.getBndryLength(), m.wrapEdges(),
                                                                                                m.getSize());
        log("Initialization - Shadow Boundaries Enabled");
} else {
        PLACES = new Places(m.getHandle(), m.getClassName(), m.getArgument(), m.getSize());
        log("Initialization - Shadow Boundaries Disabled");
}
```

REMOVE:

```
MASS.networkMap = m.getNetworkMap();
```

MThread.java

ADD:

```
else if (MASS.STATUS[0] == MASS.STATUS_EXCHANGE_BOUNDARY)
 {
    MASS.eb_exchangeBoundary();
    MASS.eb_update();
}
```

RemoteExchangeRequest.java

ADD:

```
private Boolean shadowBoundary;

public RemoteExchangeRequest(  int destIndex, int bndryIndex, Object outMsg )
{
     shadowBoundary                = true;
     destinationGlobalLinearIndex  = destIndex;
     originGlobalLinearIndex       = bndryIndex;
     inMessageIndex                = -1;
     outMessage                    = outMsg;
}

public Boolean isBoundaryRqst()  { return shadowBoundary; }
public int getBndryIndex()          { return originGlobalLinearIndex; }
```

UPDATE:

```
public RemoteExchangeRequest( int destIndex, int origIndex, int inMsgIndex, Object outMsg )
{
        shadowBoundary                = false;
        destinationGlobalLinearIndex  = destIndex;
        ...
```

Places.java

ADD:

```
41      private int         boundaryWidth;
42      private int         totalBoundaryLength;
43      private boolean     wrapEdges;
44      private Place[]     lBoundary;
45      private Place[]     rBoundary;
```

Overload Constructor:

```
74      public Places( int handle, String className, Object argument, int... size ) throws Exception
75      {
76          this( handle, className, argument, 0, false, size );
77      }

91      public Places( int handle, String className, Object argument, int boundaryWidth, boolean wrapEdge, int... size )
        throws Exception
92      {

112      this.boundaryWidth = boundaryWidth;
113      this.totalBoundaryLength = boundaryWidth;
114      this.wrapEdges = wrapEdge;
115      lBoundary = null;
116      rBoundary = null;
```

Update:

```
120      for (int elem = 0; elem < size.length; elem++)
121      {
122              if (size[elem] < 1) { throw new Exception("size value less than 1");}
123              totalLength *= size[elem]; //multiply by each element to eventually get the total length
129      }

138      // now divide work  up and store the all the information in a map
139      chunkSize = totalLength / MASS.systemSize;
140      remainder = totalLength % MASS.systemSize;
```

to

```
120      for (int elem = 0; elem < size.length; elem++)
121      {
122          if (size[elem] < 1) { throw new Exception("size value less than 1");}
123          totalLength *= size[elem]; //multiply by each element to eventually get the total length
124
125          // Calculate chunksize and remainder. Dividing the first dimension for each machine
126          if( elem == 0 ) {
127              chunkSize   *= size[elem] / MASS.systemSize;
128              remainder   *= size[elem] % MASS.systemSize;
129          }
130          else {
131              chunkSize   *= size[elem];
132              remainder   *= size[elem];
133          }
134
135          // Multiply by each element after the 1st dimension (boundaryWidth)
136          // to get total boundary size
137          if( boundaryWidth > 0 && elem > 0 ) {
138              totalBoundaryLength *= size[elem];
139          }
140      }
```

Places.java *(continued)*

ADD:

```
169         // Print Shadow Boundary information
170         if( totalBoundaryLength > 0 ) {
171             MASS.log("Boundary Shadowing : ON");
172             MASS.log("  * Boundary Width     : " + boundaryWidth);
173             MASS.log("  * Total Bndry Length : " + totalBoundaryLength);
174             if( wrapEdges )    MASS.log("  * Boundary Wrapping  : ON \n" );
175             else               MASS.log("  * Boundary Wrapping  : OFF \n" );
176         }
177         else {
178             MASS.log("Boundary Shadowing : OFF !\n");
179         }
```

REMOVE:

```
203         for(; globalLinearIndex < chunkSize; globalLinearIndex++)
204         {
205             MASS.networkMap.put(globalLinearIndex, masterRankHostName);
206         }
207
208         MASS.log("Populated master rank " + masterRankHostName +
209             " info on the network map. Total elements so far: " + MASS.networkMap.size());
210
211         // please note that # of mNode is systemSize - 1, since the master is not a part of mNodes.
212         for(int mNodeId = 0; mNodeId < MASS.systemSize - 1; mNodeId++)
213         {
214             hostName = MASS.mNodes[mNodeId].getHostName();
215             MASS.log("Populated rank " + (mNodeId + 1) + " hostname: " + hostName
216                     + " info on the network map. Total elements so far: " + MASS.networkMap.size());
217
218             int rankEndOffset = globalLinearIndex + chunkSize;
219             for(; globalLinearIndex < rankEndOffset; globalLinearIndex++)
220             {
221                 MASS.networkMap.put( globalLinearIndex, hostName);
222             }
223             if(mNodeId == MASS.systemSize - 2) // last rank gets more work
224             {
225                 for(; globalLinearIndex < totalLength; globalLinearIndex++)
226                 {
227                     MASS.networkMap.put(globalLinearIndex, hostName);
228                 }
229             }
230             MASS.nodePidMap.put(hostName, mNodeId + 1);
231         }
232
233         // DEBUG
234         System.err.println("Created network map... Total Elements now = " + MASS.networkMap.size()
235                 + " now sending information to remote nodes");
```

Update:

```
212         m.createInitializationMessage(size, arrayType, placeType, handle, className, argument, MASS.networkMap,
213             MASS.nodePidMap, dlbCount, DLBParams.HISTORY_BASED, DLBParams.WINDOW_BASED,
                DLBParams.SLOPE_BASED);
```

to

```
212         m.createInitializationMessage(size, arrayType, placeType, handle, className, argument, MASS.nodePidMap,
213             boundaryWidth, wrapEdges, dlbCount, DLBParams.HISTORY_BASED, DLBParams.WINDOW_BASED,
                DLBParams.SLOPE_BASED);
```

Places.java *(continued)*

ADD:

```
242     // Setup and Init the Left Shadow Boundary
243     int tmpIdx, k;
244     if( (MASS.myPid == 0 && wrapEdges) || MASS.myPid > 0 ) {
245         lBoundary = new Place[ totalBoundaryLength ];
246
247         for( int i = 0; i < totalBoundaryLength; i++ ) {
248             synchronized( placeInitIndex )
249             {
250                 // Note: getGlobalArrayIndex() adds offset of current machine to k when calculating
251
252                 if( MASS.myPid == 0 )   // wrapped edge
253                     k = (MASS.systemSize * chunkSize) + remainder  - totalBoundaryLength;
254                 else
255                     k = i - totalBoundaryLength;
256
257                 placeInitIndex     = getGlobalArrayIndex( k, size );
258                 placeInitSize      = size.clone();
259                 Place plc          = (Place)ctor.newInstance( argument );
260                 lBoundary[i]       = plc;
261             }
262
263             if( i == 0 )
264                 MASS.log("Left Boundary - First Location: " + (k + offSet) );
265             if( i == (totalBoundaryLength - 1) )
266                 MASS.log("Left Boundary - Last Location: " + (k + offSet) );
267         }
268     }
269
270     // Setup and Init the Right Shadow Boundary
271     if( (MASS.myPid == MASS.systemSize-1 && wrapEdges) || MASS.myPid < MASS.systemSize-1 ) {
272         rBoundary = new Place[ totalBoundaryLength ];
273
274         for( int i = 0; i < totalBoundaryLength; i++ ) {
275             synchronized( placeInitIndex )
276             {
277                 // Note: getGlobalArrayIndex() adds offset of current machine to k when calculating
278
279                 if( MASS.myPid == MASS.systemSize-1 ) // wrapped edge
280                     k = i - offSet;
281                 else
282                     k = i + chunkSize;
283
284                 placeInitIndex     = getGlobalArrayIndex( k, size );
285                 placeInitSize      = size.clone();
286                 Place plc          = (Place)ctor.newInstance( argument );
287                 rBoundary[i]       = plc;
288             }
289
290             if( i == 0 )
291                 MASS.log("Right Boundary - First Location: " + (k + offSet) );
292             if( i == (totalBoundaryLength - 1) )
293                 MASS.log("Right Boundary - Last Location: " + (k + offSet) );
294         }
295
296     }
```

Places.java *(continued)*

ADD:

```
314     /**
315        * Returns Hostname from a global index location.
316        * @param  gIdx          the global index of the location to look up
317        * @return               a string that represents the hostname of the global index loc provided
318     */
319     public  String  getHostname( int gIdx ) {
320
321             // Verify the provided info is valid
322             if( gIdx < 0  ||  gIdx >= totalLength )
323                     return null;
324
325             // Loop through each entry of the nodePidMap (node PID map)
326             for( Map.Entry< String, Integer > entry : MASS.nodePidMap.entrySet( ) )
327             {
328
329                     // Calculate the entry :  PID, first place location, and last place location
330                     int pid     = entry.getValue( );                    // pid
331                     int fLoc    = pid * chunkSize;                      // fist place location
332                     int lLoc    = ((pid+1) * chunkSize ) - 1;           // last place location
333
334                     // Add remainder to last location
335                     if( pid + 1 == MASS.systemSize )
336                             lLoc = totalLength - 1;
337
338                     if( fLoc <= gIdx  &&  gIdx  <=  lLoc )
339                             return entry.getKey();
340             }
341             return null;
342     }


586     /**
587        * New ExchangeBoundary Funcation
588        * @param  handle         the handles associated wtih a destination array
589        * @param  functionId      the identifier of a method to call
590     */
591     public  void  exchangeBoundary( int handle,  int functionId,  Vector<int[]> destinations )
592     {
593             MASS.eb_setup( this,  functionId,  destinations,  lBoundary,  rBoundary );
594             MASS.eb_exchangeBoundary();
595             MASS.eb_update();
596             if (DLBParams.WINDOW_BASED || DLBParams.HISTORY_BASED || DLBParams.SLOPE_BASED) {
597                     methodCounter++;
598                     doLoadBalancing();
599             }
600     }
601
602     /**
603        * Get Boundary Width
604        * @return       the boundary width (first dimension)
605     */
606     public  int  getBoundaryWidth( )
607     {
608             return boundaryWidth;
609     }
```

MASS.java *(continued)*

ADD:

```
55      static  final  int  STATUS_EXCHANGE_BOUNDARY      = 7;

73      // exchangeBoundary variables
74      protected static  Places     eb_places;
75      protected static  int        eb_functionId;          // function to execute
76      protected static  int[][]    eb_destinations;        // the neighbor list
77      protected static  Place[]    eb_lBoundary;           // left shadow destinations
78      protected static  Place[]    eb_rBoundary;           // right shadow destinations
```

REMOVE:

```
104   protected static HashMap<Integer, String> networkMap  =  new HashMap<Integer, String>();
```

Update:

```
756   static void ea_exchangeAll()
757   {
                ...
804             // get the host name
805             String  destHostName =     networkMap.get( globalLinearIndex );
```

to

```
805             String  destHostName =     ea_places.getHostname( globalLinearIndex );
```

MASS.java *(continued)*

ADD:

```
843     /**
844     * Exchange Boundary - Setup
845     * called by a MASS.Places object to set up the static variables for exchangeBoundary
846     * @param eb_places        the MASS.Places object to be used for the exchangeBoundary call
847     * @param ea_functionId    the function number that will be passed in to callMethod
848     * @param lBoundary        the Left  Shadow Boundary array
849     * @param rBoundary        the Right Shadow Boundary array
850     *///-------------------------------------------------------------------------------
851     static void eb_setup( Places places,  int functionId,  Vector<int[]> destinations,
852                                                            Place[] lBoundary,  Place[] rBoundary )
853     {
854             if (!INITIALIZED)   return;
855
856             // Print exchangeBoundary setup message
857             MASS.log("\nRunning ExchangeBoundary Setup (eb_setup)");
858             if( lBoundary != null )       MASS.log("  * lbSize: " + lBoundary.length );
859             else                          MASS.log("  * lbSize: NULL ");
860             if( rBoundary != null )       MASS.log("  * rbSize: " + rBoundary.length );
861             else                          MASS.log("  * rbSize: NULL ");
862
863             // Setup Global variables
864             MASS.eb_places             = places;          // the handle for this Places
865             MASS.eb_functionId         = functionId;      // callMethod function number
866
867             // Convert destinations from Vector<int[]>  to  int[][] array, then
868             // set the MASS.eb_destinations variable to the new int[][] array
869             Object[]  tmp_destinations   = destinations.toArray();
870             MASS.eb_destinations         = new int[ tmp_destinations.length ][ ];
871
872             for (int i = 0; i < tmp_destinations.length; i++) {
873                     MASS.eb_destinations[i] = (int[ ]) tmp_destinations[i];
874             }
875
876             // Set the left and right Shadow Boundary variables
877             MASS.eb_lBoundary    = lBoundary;         // Left  Shadow Boundary
878             MASS.eb_rBoundary   = rBoundary;          // Right Shadow Boudary
879
880             // Master Node Sends exchangeBoundary commands to all nodes
881             if(myPid == 0) {
882                     Message exgMsg = new Message();
883                     exgMsg.createExchangeBoundaryMessage( functionId, destinations );
884
885                     for( MNode node : mNodes ) {
886                             MASS.log("Sending eb_setup (" + exgMsg.getAction() + ") message to "
887                                     + node.getHostName() + " (pid=" + node.getPid() + ")" );
888                             node.sendMessage(exgMsg);
889                     }
890             }
891
892             // Wake-up all threads
893             synchronized (STATUS) {
894                     STATUS[0] = STATUS_EXCHANGE_BOUNDARY;
895                     STATUS.notifyAll();
896             }
897     }
```

MASS.java *(continued)*

ADD:

```
900     /**
901        * Exchange Boundary - Main Function
902     *///-------------------------------------------------------------------------------
903     static  void  eb_exchangeBoundary()
904     {
905                 // The Shadowed Place
906                 Place shdwPlace;
907
908                 // Get thread information
909                 int numThreads     = threads.size() + 1;
910                 int threadNumber   = getThreadPosition();
911
912                 // Get Boundary information
913                 int lbSize =  (eb_lBoundary == null) ? 0 : eb_lBoundary.length;
914                 int rbSize =  (eb_rBoundary == null) ? 0 : eb_rBoundary.length;
915                 int totalBndrySize = lbSize + rbSize;
916                 int maxBndrySize   = (lbSize > rbSize) ? lbSize : rbSize;
917
918                 // Verify system has been initialized and at least one of the shadow boundaries exists
919                 if ( !INITIALIZED || totalBndrySize == 0 ) return;
920
921                 // Loop through all of the shandow boundary locations (left and/or right)
922                 int bndryIdx = threadNumber;
923                 while( bndryIdx < totalBndrySize )
924                 {
925                         // Get shadow boundary place
926                         if( bndryIdx < lbSize )         shdwPlace = eb_lBoundary[ bndryIdx ];                   // Left Boundary
927                         else                            shdwPlace = eb_rBoundary[ bndryIdx - lbSize ];          // Right Boundary
928
929                         // Get shadow global linear index location and the destination hostname
930                         int shdwGlobalLinearIdx =
931                                 Places.getGlobalLinearIndexFromGlobalArrayIndex( shdwPlace.index, eb_places.size() );
932                         String destHostName = eb_places.getHostname( shdwGlobalLinearIdx );
933
934                         // Create a new request
935                         RemoteExchangeRequest request = new RemoteExchangeRequest( shdwGlobalLinearIdx, bndryIdx, null );
936
937                         synchronized(exchangeAllRequestMap)
938                         {
939                            if(exchangeAllRequestMap.get(destHostName) == null) {
940                                 ArrayList<RemoteExchangeRequest> requests = new ArrayList<RemoteExchangeRequest>();
941                                 requests.add(request);
942                                 exchangeAllRequestMap.put(destHostName, requests);
943                            } else {
944                                 exchangeAllRequestMap.get(destHostName).add(request);
945                            }
946                         }
947                         //MASS.log("ExchangeBoundary 'exchangeAllRequestMap' size : " + exchangeAllRequestMap.size());
948                         bndryIdx += numThreads;
949                 }
950
951                 // Process the Remote exchange requests with the remote nodes
952                 barrier();
953                 processRemoteExchangeRequest( );
954                 barrier();
955     }
```

MASS.java *(continued)*

ADD:

```
958     /**
959        * Exchange Boundary - Update Function
960     *///---------------------------------------------------------------------------
961     static  void  eb_update()
962     {
963               if ( !INITIALIZED ) return;
964               MASS.log("Starting Exchange Boundary Update...");
965
966               // Create Places Iterator for the places assigned to this thread
967               int[] thrdRange                 = getLocalRange( eb_places );
968               Places.Iterator origin_iter  = eb_places.iterator( thrdRange );
969
970               if ( origin_iter != null )                    // null when not enough MASS.Place objects for thread
971               {
972                         int[ ] size = eb_places.size( );
973                         Place origin;                              // for caller MASS.Place
974                         Place dest;                                // for callee MASS.Place
975                         int in_msgs_len = eb_destinations.length;
976
977                         while ( origin_iter.hasNext( ) )              // go through this thread's range
978                         {
979                                   origin = origin_iter.next( );
980                                   if ( origin.inMessages == null || origin.inMessages.length != in_msgs_len ) {
981                                             origin.inMessages = new Object[ in_msgs_len ];
982                                   }
983                                   int inMessagesIndex = 0;
984
985                                   // Loop through all of the destinations (neighbors)
986                                   for ( int dest_i = 0;  dest_i < in_msgs_len;  dest_i++ ) {
987
988                                      // fill dest_coords
989                                      int[] neighborCoord =
990                                             Places.getGlobalNeighborArrayIndex( origin.index, eb_destinations[ dest_i ], size );
991
992                                      // If a Valid Destination, update destination information in inMessages
993                                      if ( neighborCoord[0] != -1 ) {
994
995                                             int globalLinearIndex =
996                                                       Places.getGlobalLinearIndexFromGlobalArrayIndex( neighborCoord, size );
997                                             int destinationLocalLinearIndex =
998                                                       Places.getLocalLinearIndexFromGlobalLinearIndex( globalLinearIndex );
999
1000                                            // On Destination Machine
1001                                            if(  ( 0 <= destinationLocalLinearIndex )  &&
1002                                                ( destinationLocalLinearIndex < eb_places.length() ) )
1003                                            {
1004                                                      dest = eb_places.get( destinationLocalLinearIndex );
1005                                                      origin.inMessages[ inMessagesIndex ] =
                                                                dest.callMethod(eb_functionId, origin.outMessages);
1006                                            }
1007
1008                                            // Left Shadow Boundary
1009                                            else if(    ( destinationLocalLinearIndex < 0 )  &&
1010                                                      ( eb_lBoundary.length + destinationLocalLinearIndex  < eb_lBoundary.length ) )
1011                                            {
1012                                                      dest = eb_lBoundary[ eb_lBoundary.length + destinationLocalLinearIndex ];
```

MASS.java *(continued)*

```
1013                                            origin.inMessages[ inMessagesIndex ] = dest.outMessages;
1014                                    }
1015
1016                                    // Right Shadow Boundary
1017                                    else if(    ( destinationLocalLinearIndex >= eb_places.length() ) &&
1018                                            ( destinationLocalLinearIndex - eb_places.length() ) < eb_rBoundary.length )
1019                                    {
1020                                            dest = eb_rBoundary[ destinationLocalLinearIndex - eb_places.length() ];
1021                                            origin.inMessages[ inMessagesIndex ] = dest.outMessages;
1022                                    }
1023
1024                                    // Error - not found
1025                                    else {
1026                                            origin.inMessages[ inMessagesIndex ] = null;
1027                                    }
1028                            }
1029
1030                            // Invalid Destination, coordinates outside this MASS.Places
1031                            else {
1032                                    origin.inMessages[ inMessagesIndex ] = null;
1033                            }
1034
1035                            inMessagesIndex++;
1036                        } // end of destination for loop
1037                    }
1038                    MASS.log("Exchange Boundary Update Complete!");
1039            }
1040            barrier();
1041    }
```

Update:

```
1185    static ArrayList<RemoteExchangeRequest>  doRemoteExchangeAll( ArrayList<RemoteExchangeRequest> requestList )
1186    {
1187            Place destination;
1188            ArrayList<RemoteExchangeRequest> retList = new ArrayList<RemoteExchangeRequest>();
1189
1190            for(RemoteExchangeRequest request : requestList)
1191            {
1192                destination = ea_places.get(
1193                                Places.getLocalLinearIndexFromGlobalLinearIndex( request.getDestinationGlobalLinearIndex() ));
1194                Object returnMessage = destination.callMethod( ea_functionId, request.getOutMessage() );
1195                RemoteExchangeRequest returnReq = new RemoteExchangeRequest( request.getDestinationGlobalLinearIndex(),
1196                                                                             request.getOriginGlobalLinearIndex(),
1197                                                                             request.getInMessageIndex(),
1198                                                                             returnMessage);
1199               retList.add(returnReq);
1200            }
```

to

```
1186    static ArrayList<RemoteExchangeRequest>  doRemoteExchangeAll( ArrayList<RemoteExchangeRequest> requestList )
1187    {
1188            Place destination;
1189            RemoteExchangeRequest returnReq;
1190            ArrayList<RemoteExchangeRequest> retList = new ArrayList<RemoteExchangeRequest>();
1191
1192            for( RemoteExchangeRequest request : requestList )
1193            {
1194                    int tmpDestGlbLinIdx = request.getDestinationGlobalLinearIndex();
1195                    int tmpDestLocLinIdx = Places.getLocalLinearIndexFromGlobalLinearIndex( tmpDestGlbLinIdx );
1196
1197                    // If request is a exchangeBoundary request
1198                    // ----------------------------------------
1199                    if( request.isBoundaryRqst() ) {
1200
1201                            if( eb_places == null ) { MASS.log("***  ERROR: eb_places == null  ***"); System.exit(-1); }
1202                            destination                 = eb_places.get( tmpDestLocLinIdx );
1203                            Object returnMessage        = destination.callMethod( eb_functionId, request.getOutMessage() );
1204
1205                            // Create a new return request to send back the return values
1206                            returnReq      = new RemoteExchangeRequest(         request.getDestinationGlobalLinearIndex(),
1207                                                                                request.getOriginGlobalLinearIndex(),
1208                                                                                returnMessage  );
1209                    }
1210
1211                    // If request is an exchangeAll request
1212                    // ----------------------------------------
1213                    else {
1214                            destination                 = ea_places.get( tmpDestLocLinIdx );
1215                            Object returnMessage        = destination.callMethod( ea_functionId, request.getOutMessage() );
1216
1217                            // Create a new return request to send back the return values
1218                            returnReq      = new RemoteExchangeRequest(         request.getDestinationGlobalLinearIndex(),
1219                                                                                request.getOriginGlobalLinearIndex(),
1220                                                                                request.getInMessageIndex(),
1221                                                                                returnMessage  );
1222                    }
1223                    retList.add(returnReq);
```

Update:

```
1232    static void updateInMessages( ArrayList<RemoteExchangeRequest> requestList )
1233    {
1234            Place origin;
1235            ArrayList<RemoteExchangeRequest> retList = new ArrayList<RemoteExchangeRequest>();
1236
1237            for(RemoteExchangeRequest request : requestList)
1238            {
1239                    origin = ea_places.get(
                                Places.getLocalLinearIndexFromGlobalLinearIndex(request.getOriginGlobalLinearIndex()));
1240                    origin.inMessages[request.getInMessageIndex()] = request.getOutMessage();
1241            }
1242    }
```

to

```
1232    static void updateInMessages( ArrayList<RemoteExchangeRequest> requestList )
1233    {
1234            Place origin;
1235            ArrayList<RemoteExchangeRequest> retList = new ArrayList<RemoteExchangeRequest>();
1236
1237            for(RemoteExchangeRequest request : requestList)
1238            {
1239
1240                    // If request is a exchangeBoundary request
1241                    // ----------------------------------------
1242                    if( request.isBoundaryRqst() ) {
1243
1244                            int idx    = request.getBndryIndex();
1245                            int lbSize = (eb_lBoundary == null) ? 0 : eb_lBoundary.length;
1246
1247                            // Determine the shadow boundary location (left or right), then retrieve it
1248                            if( idx < lbSize )    origin = eb_lBoundary[ idx ];
1249                            else                  origin = eb_rBoundary[ idx - lbSize ];
1250
1251                            // Update the shadow boundary outMessage variable with the retrieved value
1252                            origin.outMessages = request.getOutMessage();
1253                    }
1254
1255                    // If request is an exchangeAll request
1256                    // ----------------------------------------
1257                    else {
1258                            origin = ea_places.get(
1259                                    Places.getLocalLinearIndexFromGlobalLinearIndex(request.getOriginGlobalLinearIndex()));
1260                            origin.inMessages[request.getInMessageIndex()] = request.getOutMessage();
1261                    }
1262            }
1263    }
```

Update:

```
2296    private static void agentsManageAllPerThreadKillOrStartMigrate()
2297    {
        ...
2343                    String destHostName = networkMap.get(globalLinearIndex);
```

to

```
2343                    String destHostName = ( ea_places != null )   ?  eb_places.getHostname(globalLinearIndex)
2344                                                                  :  ea_places.getHostname(globalLinearIndex);
```

NO CHANGES

Agent.java
Agents.java
ExchangeHelper.java
MNode.java
Place.java
RemoteAgentRequest.java
Utilities.java