Motivation to a Deadlock Detection in Mobile Agents with Pseudo-Code

Rashmi Priya[1(✉)] and R. Belwal[2]

[1] TMU, Moradabad, India
rashmi.slg@gmail.com
[2] AIT, Haldwani, India
r_belwal@rediffmail.com

Abstract. The solution presented locates locality of reference during the deadlock detection process by migrating detector agents to query multiple blocked agents. To message each blocked agent individually and gather their responses at the shadow agent itself is an alternative to this single migration. The pseudo code provides a context for the solution and insight into the responsibilities and activities performed by each entity.

Keywords: Deadlock · Agents · Pseudo code · Detector · Entity

1 Introduction

As presented in the previous section traditional distributed solutions commonly have fault and location assumptions that make them unsuitable for mobile agent systems. To solve this problem, mobile agent specific solutions are required. The properties of the presented deadlock detection algorithm illustrate how it is a fully adapted mobile agent solution. The presented technique is fault tolerant and robust. Lost agents or messages during the deadlock detection process do not represent a critical failure. This fault tolerance is due to three properties of the algorithm: the autonomous nature of the agents, the periodic nature of the detection process and the copying of deadlock information. Shadow deadlock detection and consumer agents execute asynchronously. They do not depend on continual communication during the deadlock detection process. The algorithm is designed around incremental construction of the global wait-for graph. Finally, therefore if a portion of the graph is lost, the next update will recover that information. Hence copying of the partial wait-for graph into deadlock detection agents make the loss or failure of a particular deadlock detection agent trivial and has no impact on the detection process, outside of slowing the process. Additional safeguards can be built into the agent hosts. such as agent crash detection, to improve fault tolerance.

1.1 Algorithm Motivation and Agent Properties

By limiting the number of messages that would be required in other solutions the detector migration reduces network load. It is difficult to compare the network load of

S.C. Satapathy and A. Joshi (eds.), *Information and Communication Technology for Intelligent Systems (ICTIS 2017) - Volume 1*, Smart Innovation, Systems and Technologies 83, DOI 10.1007/978-3-319-63673-3_14

this mobile agent solution to that generated in traditional distributed deadlock detection solutions due to the significantly different paradigm and properties of the environment. This is due to the parallel/distributed nature of the technique, which enforces the lack of a central point of messaging and coordination. This reduces the risk of flash congestion and allows the technique to handle deadlock involving many blocked agents.

The load is spread across many host environments, if the network load of the presented solution is considered as a whole. Additionally, network organization independence is guaranteed through a clear separation of mobile agents from the mechanics of routing and migration, the agents are not aware of the number of hosts in the mobile agent system and do not have explicit knowledge of resource locations. It should be noted that even though the solution is network independent, the topology is static once the algorithm begins. If the topology is allowed to change, a dynamic topology update protocol must execute in the background to provide new routes to the hosts.

A common use of mobile agents is to encapsulate complex protocols and interactions [6]. This technique uses the combination of shadow agents and deadlock detection agents to encapsulate a complex series of probes, interactions and acknowledgments. Additionally, these protocols are isolated from the consumer agent; therefore, can be easily modified and upgraded. The deadlock detection phase could be implemented as remote procedure calls or another form of distributed programming, but would require network organization assumptions and the continual exchange of messages. Detector and shadow agents carry out their deadlock detection tasks in an asynchronous manner. They coordinate their efforts in defined ways, but are able to keep working without regular contact and do not require constant supervision while carrying out tasks. This asynchronous and autonomous operation contributes to the previously discussed fault tolerance. For example, the combination of consumer, shadow and detector agents adapt to their environment to solve deadlock situations, shadow agents react independently to changing network conditions and the state of their target consumer agent to initiate the deadlock detection processing. Similarly, the separation of the implementation from facilities specific to a particular mobile agent system or operating system detector agents can react to network failures or the requests of other agents while gathering global wait-for graph information allows the solution to execute in a heterogeneous environment. Moreover, the separation of replica and detector agents from the consuming agents they monitor, allows them to be adapted to many different environments without (or with minor) modifications to the entities performing the work.

2 Deadlock Detection Pseudocode

This pseudo code provides a context for the solution and insight into the responsibilities and activities performed by each entity. This section presents pseudo-code of each element that plays a significant role in the presented solution. First, pseudo-code for the consumer, shadow and detection agent is presented. Finally, code for the mobile agent environment is presented.

2.1 Agent A

```
public class AgentA extends MobileAgent
{
public AgentA( String int heartbeat )
{
state = IDLE;
}
public void run()
{
while( true )
{
messages = getMessagesFromBlackboard
(agentId);
processMessages( messages );
switch ( state )
{
case IDLE:
case WAITING:
break;
case MOVING:
if( currentHost is not targetEnvironment )
{
postRouteRequest( targetEnvironment ) ;
{
else
{
state = IDLE;
{
break;
{
AND state
private void lockResource( String locktype,
String resourceName )
{
if( lockType equals  “exclusive” )
{
get resource manager;
if resourceManager.lockResource(
resourceName,
lockType ) succeeds
{
if (resourceName equals
blockedResourceName )
state = IDLE;
{
sleep ( heartbeatDelay ) ;
{
private Vector processMessages( messages )
{
while ( more Messages )
if message was accepted remove from list;
return unprocessed messages;
{
private boolean processMessages
BlackboardEntry msg )
if ( message equals amove" AND state is IDLE
)
{
targetEnvironment = get target from message;
state = MOVING;
postBlackboardMsg( "route”,
targetEnvironment ) ;
{
else if( message eqyals "lockw AND
state is IDLE OR WAITING )
{
extract lock type and resource from message ;
lockResource( lockType, resource );
if( message equals
{
extract resource from message;
unlockResource( resource );
{
{
else
state = WAITING;
blockedResourceName = resourceName;
postBlackboardMessage(“agentBlock”,
resourceName );
private void unlockResource( String
resourceName )
{
get Resource Manager
resourceManager.unlockResource(resourceNa
me);
}}
```

2.2 Agent B

```
public class ReplicaAgent extends
MobileAgent
(
public ReplicaAgent ( String id, String
targetAgent, int heartbeat )
{
state = IDLE;
targetAgentName = targetAgent;
reset locked resource list
reset detection info table
reset detector agent
{
public void run()
while( true )
{
if ( state is not MOVING )
{
messages = getMessagesFrom BlackBoard();
messages =processMessages(messages);
{
switch ( state)
{
case IDLE:
break;
get~messagesFromBlackboard();
processMessages( messages );
case MOVING:
if( currentHost is not targetEnviroment ) )
{
routeRequest( targetEnvironment );
{
else
{
state = IDLE;
}
break;
else if( message equals ablockedm ) )
{
blockedTarget( attachment #l );
retVal = true;
{
else if( message equals wblocked ) )
{
unblockedTarget ( ) ;
retVal = true;
{
}
sleep (heartbeatDelay) ;
}
}
private Vector processMessages( messages )
{
while( more Messages )
{
processMessage( currentMessage );
if message processed remove from list;
 {
return unprocessed messages;
{
private boolean processMessage(
BlackboardEntry msg )
{
Vector attachments = msg.getAttachments();
if( message equals "move" AND state is IDLE)
{
String target = extract target from message;
targetEnvironment = target;
state = MOVING;
{
else if( message equals “addLock” )
{
addlock( attachment #l,
attachent #2,
attachment #3,
attachment #4 );
retVal = true;
else if( message equals  “removeLock” )
{
removelock ( attachment #I, attachment #2 ) ;
state = IDLE;
retVal = true;
{
private void blockedTarget( Agent
blockedAgent,
String (resourceName )
{
State1 = TARGETBLOCKED;
owner = query host environment for owner of
resource;
blockedResourceName = resourceName;
localAgents = query host environment about
local agents;
```

```
else if( message equals “deadlockReport” ) )
{
processReturnOfDetector( attachment#1,
attachment #2, attachment #3 ) ;
retVal = true;
{
else if( message equals “deadlockInfoRequest”
) 1
retVal = true;
else
{
retVal = super.processMessage( msg );
{
{
public void exit ()
{
if( detector )
{
Remove ( detector from host environment);
{
super. exit () ;
private void addlock( String
environment,String resource,String owner,int
priority1 )
if( resource not already locked )
{
new r e s o u r c e I n f o ( env, res, owner, p r
i o r i t y1 ) ;
s t o r e resourceInfo i n locked resource list
}
}
Private void removeLock( S t r i n g env1, S t
r i n g resourceName1 )
{
i f ( resourceName1 is i n locked resource list )
{
remove resource from locked resource list
{
{
private void unblockedTarget1( )
{
state = IDLE;
{
if ( owner in localAgents )
Table .put ( targetAgentName, new
DetectionInfo( .. ) );
Detector1 = new DetectorAgentO;
postBlackboardMsg( d e t e c t o r );
postBlacKboardMsg( buildDetectorLocks () ) ;
numOfDetectionStarts++;
1astDetectionStartTirne = current time ;
{
{
p r i v a t e void processReturnOfDetector(
DetectorAgent agent )
switch( state )
{
case TARGET-BLOCKED :
if ( checkForDeadock(
agent.getDetectionTables()
reset detectionInfoTable;
{
case IDLE:
removeDetector () ;
(
switch ( state )
(
case TARGET-BLOCKED:
postBlackboardMsg( " s t a r t " ,
buildDetectorLocks() );
numOfDetectionStarts++; )
1astDetectionStartTime = currenttime ;
break;
case WAITING,FOR UNLOCK:
break;
{
{
p r i v a t e boolean checkForDeadlock( Vector
detectionTableList )
{
while ( detect Table List has more entries )
{
detectionTable = current detection table ;
agent List = get agent list from det. ;
while( agent List has more entries )
{
detection Info = detection info ;
```

```
if ( current agent name equals
targetAgentName )
{
deadlockFound = True;
{
{
{
return deadlockFound;
{
p r i v a t e void resolveDeadlock(
DetectorAgent agent )
{
if( state is TARGETBLOCKED )
{
cycleList = findElementsInCycle(
detectionInfoTable ) ;
while( cycleList has more elements )
{
1ockToBreak = lowest priority resource;
{
i f ( lockToBreak equals resource we are
blocked on )
{
postBlackboardMsg( "unlock",LockToBreak );
state = WAITING-FOR-UNLOCK;
{
{
private Vector findcycle ( Hashtable
detectionInfoTable )
{
Vector cycleVector = new Vector ( ) ;
cyclevector add( resource we are blocked on ) ;
info = find entry in detectionInfoTable
while( current entry agent name not equal to
our target agent )
{
info = find entry in detectionInfoTable ;
cyclevector add( info );
{
return cyclevector;
{
private void checkForDetectorDeath()
{
Date currentTime = current time;
BlackboardEntry msg;
if( num0fDetectionStarts > 0)
postBlackboardMsg ( "inject", detector ) ;
if( state is TARGET-BLOCKED )
{
postBlackboardMsg( "start",
buildDetectorLocks() ;
{
else if( state is WAITING-FOR-UNLOCK )
resolveDeadlock( detector.getIdentifier());
detector.getToken() );
{
1astDetectionStartTime = current time;
}}}}
```

2.3 Agent C

```
public class AgentC extends MobileAgent
{
public  AgentC ( String id, int heartbeat,
ShadowAgent parent )
{
reset detection Table List;
reset resources To Vist;
reset targetEnvironment;
reset targetResource;
state = IDLE;
set parent = parent;
{
public void run ()
{
while ( true )
{
if ( state is not MOVING )
{
messages = getMessagesfromBlackboard () ;
messages = processMessages( messages );

}
switch ( state)
getMessagesfromBlackboard () ;
processMessages( messages );
(
case IDLE:
break;
case MOVING:
if( currentHost is not targetEnvironment ) )
{
else
{
state = CHECKING-LOCKS;
{
break;
 if( current Host is not targetEnvironment )
{
```

```
if( host.unlockResource( targetResource,
agentToNotify )
{
( state = RETURN-FROM-UNLOCK);
{
else
state = IDLE;
{
case RETURNRNFROM, the LOCK:
if( currentHost is not startingEnvironment )
(
Shadow,removeLock( targetEnvironment,
targetResource );
state = IDLE;
{
case DONE:
if( currenthost is not startingEnvironment )

(
state = REPORT-RESULTS;
)
case CHECKING-LOCKS:
checklocks ( ) ;
break;
case REPORT-RESULTS:
postMessageToBlackboard( shadowÀgent,
deadlockInfo );
break;
}
sleep( heartbeatDelay ) ;
private Vector processMessages( messages )
{
while ( more Messages )
{
processMessage( currentMessage );
if message processed remove from list;
{
return unprocessed messages;
{
private boolean processMessage(
BlackboardEntry rnsg )
{
attachments = msg,getAttachments();
if( message equals  “startW ”)
{
startDetection ( (Vector) attachment  ;
retVal = true;
else if( message equals "unlock" ) {
startunlock ( attachment #1,attachment #2,
attachent #3 ) ;
retVal = true;
message ;
deadlockRequestResponse ( attachment #l ) ;
retVal = true;
{
else
{
super.processMessage( msg 1;
{
return retVal;
{
private void startDetection( resources )
( setVisitlist( resources ));
targetEnvironment( entry.getEnvlame() };
targetResource ( entry. getResName () ) ;
detectionTablelist( new Vector 1);
start ingEnvironment (getHost ().getName ());
state ( MOVING ) ;
{
private void checklocks ()
{
while( shadowlist has more elements )
count expected responses;
{
if( expected responses > 0)
(
state = WAITING-FOR-RESPONSE;
{
else
{
findNewTarget () ;
}
private void deadlockRequestResponse(
newTable )
{
shadowList = query current host for agents
blocked on
the resource we are visiting;
expectedResponses--;
detectionTablelist.add( newTable );
if( all expectertesponses received )
{
findNewTarget 0 ;
}
}
private void findNewTarget0
{
if( more resource to visit )
get next resource;
targetEnvironment = entry.getEnvName();
targetResource = entry.getResName();
{
else
{
targetEnvironment = startingEnvironment;
state = DONE;

}}}
```

2.4 Host Environment

```
public class AgentEnvironment extends
Thread
public AgentEnvironment( String name, int id,
int (1oggingLevel )
{
resourceManager = new ResourceManagerO;
topologyManager = new TopologyManager();
reset agentTable;
reset messageBoard;
reset blockedAgentTable;
globalIdentifier = id;
state = PROCESSING;
}
public void run ()
(
while( true )
{
checkErrorMessages () ;
updateRoutes () ;
sleep( 1000 );
}
)
public synchronized void agentEnter( Agent
newAgent )
{
if( state is PROCESSING )
{
agentTable.put( newAgent ) ;
newAgent.enter0;
}
}
private synchronized void agentExit( Agent
1eavlngAgent
{
if ( state is PROCESSING )
(
1eavingAgent. exit ( ) ;
)}
private void agentBlock( Agent blockedAgent,
String resourceName)
(
replica = find replica agent for blockedAgent;
if ( shadow found )
{
postBlackboardMsg( "blockedAgent",
resourceName ) ;
private void checkForMessages()
{
get messages from blackboard;
while ( more messages)
{
processMessage( current message );
}
if ( env is not null )
}
private void processMessage(
BlackboardEntry msg
attachments = rnsg.getAttachments0;
if( message equals “pause” ) )
state( PAUSED ) ;

else if( message equals wresumen ) )
(
state( PROCESSING ) ;
message equals
(
agentBlock ( msg . getAgent Id () , attachment
t1 ,attachment t2);
if( message equals 0)
{
routeRequest(msg.getAgentId(), attachment # l
,attachment #2);
{
else if (( message equals *inject))
this.injectAgent( attachment)

else if( message equals "remove" ) )
{
removeAgent ( attachrnent #l ) ;
}
}
private boolean routeRequest( String
movingAgent, EnvironmentToken token,
String targetEnv )
{
if( state() is PROCESSING )
{
movingAgent = get moving agent from agent
tables;
(
return true;
)
if( check for shadow information in the token )
shadowAgentId = get shadow name from
token;
If ( check for shadow agent in agent tables )
{
shadow = get shadow agent from agent tables;
}
else
{
retVal = f alse;
}}
if ( retVal is true )
(
AgentEnvironment env = request route from
topologyManager;
(
```

3 Conclusion

The presented algorithm is designed with the unique properties and challenges of mobile agent systems as a motivating factor. As a result, the solution has some of the properties and features that are commonly found in mobile agent implementations. This section lists the properties of the proposed algorithm which make it a mobile agent solution. The solution is network organization independent. The algorithm makes no assumptions concerning network topology (i.e., ring). The number of hosts or node locations to support the solution. Resource-based routing and tracking of the nodes visited by a particular agent eliminate the need for explicit topology knowledge.

References

1. Walsh, T., Paciorek, N., Wong, D.: Security and reliability in Concordia. In: Mobility: Processes, Computer and Agents. Addison-Wesley, Reading (1999)
2. Mitsubishi Electric Information Technology Center: Concordia—Java Mobile Agent Technology. World Wide Web, January 2000. http://www.meitcacom/HSUProjects/Concordia/
3. University of Stuttgart: The Home of the Mole. World Wide Web, September 2000. http://mole.infonnatik.uni-stuttgart.de/
4. University of Tromse and Cornell University: TACOMA—Operating System Support for Mobile Agents. World Wide Web, August 1999. http://www.tacoma.cs.uit.no/
5. ObjectSpace, Inc.: ObjectSpace voyager core package technical overview. In: Mobility: Processes, Computer and Agents. Addison-Wesley, Reading (1999)
6. ObjectSpace, Inc.: ObjectSpace Product Information: Voyager. World Wide Web, September 2000. http://www.objectspace.com/products/voyager/
7. Fachbereich lnfonnatik and Johann-Wolfgang-Goethe-Universitaet Frankfurt: "ffMain". World Wide Web, February 2000. http://www.tm.informatik.uni.frankfurt.de/Projekte/MN
8. University of Geneva: The Messenger Project. World Wide Web, December 1997. http://cui.unige.ch/tios/msgr/
9. General Magic, Inc.: Odyssey. World Wide Web, November 2000. http://www.genmagic.com/
10. University of Modena: MARS (Mobile Agent Reactive Space). World Wide Web, October 2000. http://sirio.dsi.unimo.it/MOON/MARS/index.html