5. SERVICE REPLICA MANAGEMENT

5.1 INTRODUCTION

A Distributed UDDI system is similar to any Distributed Application or a Distributed Database. In a Distributed Database the collection of data is kept in different servers each located at different geographical locations and connected through a networking infrastructure. The major problem of a Distributed Database is the Consistency Management and the Replica Management. When a database of identical data is kept in different server in different locations through data replication, access becomes easier. All data retrieval can be done from a nearby server than accessing it from a remote server which ensures the data availability to the requester in shorter time. But the problem is the replica management. Proper ways and methodologies only will ensure the consistency of replicas kept in different servers. Similarly, the services information, which is replicated in different UDDI nodes should be consistent across all nodes of the DUDDI system, wherever the services information have been replicated. So, Replica management in DUDDI system assumes greater importance. In this Chapter, methodologies have been proposed for Service Replica Management. The proposed methodology has been well studied in the already established DUDDI system (Chapter 4) and the simulated data has been analyzed and assessed for its performance through the set of performance attributes identified (Section 3.3.3.3 and Section 3.3.3.5).

In this chapter, a methodology has been proposed for an efficient node search in the DST structured P2P based DUDDI system (Section 5.2). This creates a base to understand how a particular service’s information is accessed from a particular node, which stores the replica of that information. In Section 5.3, the proposed methodologies for the Replica Management and the Read and write operations in the DUDDI system are discussed.
5.2 NODE SEARCH IN LARGE SCALE DST STRUCTURED P2P DUDDI

Node search in a large scale DST Structured P2P based DUDDI is a challenging assignment for retrieving the service information. A systematic approach using DST in the P2P based DUDDI system and the proposed methodology, as given in Figure 5.1, ensures an efficient node search.

The proposed Node Search Methodology consists of five procedures, which are \texttt{Request()}, \texttt{toAllHeads()}, \texttt{Found()}, \texttt{Reply()} and \texttt{Receive()}. Each procedure is meant for a specific task in the overall node search process and the details of about each procedure is discussed as given below.

\textbf{Request():} Procedure \texttt{Request()} is invoked by any node, like for example a Service Consumer node in the DUDDI system which wants to identify a Service Provider node for communication. There are two arguments to be passed to this procedure when invoked. One, the source node \textquote{v} and two, the destination node \textquote{d}. A \texttt{doFind} message is created and sent to the Head Node of the source \textquote{v} along with the details of the source and the destination.

\textbf{toAllHeads():} A node \textquote{v}, which should be a Head Node, invokes this procedure when it receives the \texttt{Request} message to propagate a message to all other Head Nodes. The Head Node \textquote{v}, which invokes this procedure communicates the received message to all the other Head Nodes of the system through the information available in the \texttt{HN.ARR} array of \textquote{v}. 


Figure 5.1 Procedures for Node Search using DST Structures
**Found():** When a Head Node (HN₁) intends to reach a destination node ‘d’ and requests the other Head Nodes, this is invoked by a Head Node (HN₂) which hold the requested destination node ‘d’ and creates the foundDest message with the fields ‘Dest’ and ‘Head’ set as the id of the destination node ‘d’ and id of the Head Node (HN₂). The foundDest message is forwarded to the Head Node (HN₁) by the Head Node (HN₂) by invoking the Procedure Reply().

**Receive():** Procedure Receive() is invoked by a node ‘v’, which may be either a Head Node or Leaf Node. Node ‘v’ invokes this procedure when it receives any message and the received message should be any one of the three variants, which are a doFind message or a toallHeads message or the foundDest message.

If the received message by node ‘v’ is a doFind message, which is to search and find a particular node in the system, any one of the following occurs as given in the Case 1 and Case 2 as follows.

**Case-1:** If the receiving node ‘v’ is one of the Head Nodes the system, then the ‘Head’ field of the message is checked. If the ‘Head’ field of the message is the ‘id’ of the node the same Head Node, ‘v’, then the procedure toAllHeads() is invoked.

**Case-2:** If case-1 fails (either node ‘v’ is not a Head Node or the ‘Head’ field is not the ‘id’ of ‘v’), then the node ‘v’ forwards the message to the node with ‘id’ in the ‘Head’ field of the message.

If a node ‘v’ of the systems receives a toallHeads message, which is meant for all the Head Nodes of the system, then it could be to search and identify a destination node under its supervision. To find out whether the requested destination node is under its supervision or not the ‘d’ field of the message is to be
verified. After receiving a *toallHeads* message any one of the two cases, Case 1 or Case 2 would happen which are discussed as given below.

**Case-1:** If the node ‘v’ is a Head Node and ‘Head2’ field of the message is ‘id’ of the node ‘v’, then node ‘v’ retrieves the ‘d’ field (destination node) of the message and matches it with the LN entry in its *LN_ARR* array. The matching process leads to either Case-1a or Case-1b.

**Case-1a:** If the ‘d’ field in the message matches with the LN entry of array *LN_ARR* in ‘v’, which means that the search node ‘d’ is under the current Head Node ‘v’. Then the node ‘v’ invokes the procedure *Found()* to intimate the requester Head Node.

**Case-1b:** If there are no matches found in *LN_ARR* array of the Head Node ‘v’, then it means that the searched node ‘d’ is not under this Head Node ‘v’ and the the received *toallHeads* message gets discarded.

**Case-2:** If case-1 fails (it may be either the case that the receiving node is not a Head Node or the ‘Head2’ field of the received message is not the ‘id’ of the node ‘v’), then the node ‘v’ forwards the message to the node with the ‘id’ in the ‘Head2’ field of the message.

If a node ‘v’ of the system receives a *foundDest* message, which is meant to communicate that the destination node is found. The node ‘v’ which receives the *foundDest* message decides based on whether the node ‘v’ is a Head Node or Leaf Node and any one of the following three cases would happen as explained as follows.

**Case-1:** If the node ‘v’ is a Head Node and the ‘Head1’ field of the message is ‘id’ of the node ‘v’, then the procedure *Reply()* is invoked.

**Case-2:** If the node ‘v’ is a Leaf Node and the ‘s’ field of the message is ‘id’ of the node ‘v’, then it shows that the destination ‘d’ is found. The node ‘v’
retrieves the ‘Head1’, ‘Head2’ fields of the message which it uses for the communication with the destination ‘d’ through destination Head Node ‘Head2’.

**Case-3:** If both case-1 and case-2 fails (which means that the received message is not intended for the node ‘v’), then the node ‘v’ forwards the message to the node with ‘id’ in the ‘s’ field of the message.

The Proposed Node search methodology uses the details stored by every Head Node regarding its Leaf Nodes. This approach makes the node/resource search operation in a large scale P2P based DUDDI system more efficient and economic with very minimum number of message passes (Chapter 6) and reduced routing table entries in every node of the system.

**Definition 1:** Let $P_i$ be a node in the DUDDI system. Then the number of routing table entries required for the node $P_i$ to route any message it receives is $n(P_i(\text{RT_entry}))$, which is given as,

$$n(P_i(\text{RT_entry})) = n(HN) + \{n(HN_i(\text{LN_ARR})) \cup r(P_i)\} \quad - (5.1)$$

where,

- $n(P_i(\text{RT_entry}))$ are the number of routing table entries required for node $P_i$ to route any message it receives.
- $n(HN)$ are the number of HNs in the Network.
- $n(HN_i(\text{LN_ARR}))$ are the number LNs in the $LN\_ARR$ of HN of $P_i$.
- $r(P_i)$ are the number of nodes adjacent to $P_i$.

As per Equation 5.1, the number of routing table entries $n(P_i(\text{RT_entry}))$ required for node $P_i$ to route any message it receives is the sum of the number of Head Nodes (HNs) in the system and combination (without duplicate) of number of
Leaf Nodes (LNs) in the \textit{LN\_ARR} of HN of $P_i$ and number of nodes adjacent to node $P_i$.

5.3 DST IN REPLICA MANAGEMENT

Replica Management refers to the process of maintaining the published service information across different SRL nodes (the Head Nodes) in a consistent fashion with appropriate Service Read and Write operations performed by the Service Providers and Consumers of the proposed DUDDI system. In this research work, the proposed model performs the Read/write operation in 3 phases, which basically consists of three rounds of message exchanges and they are 1) A lock request/its reply, 2) Read/Write operations and their Acknowledgment and 3) Commit and Lock Release and its Acknowledgment. The methodology for Read and Write operations is described in the following Section.

5.3.1 Initialization

\textit{Local read Lock (LL) and Global write Lock (GL)}: LL and GL are the flags which can be set or reset and be maintained by each Head Node to avoid the replica becoming inconsistent due to concurrent multiple requests for Read and Write operations. The values of LL and GL are to be reset at start.

\textit{Priority Number (PN)}: PN is a constant, randomly assigned for each Head Node at the time of creating the Head Node. PN will be useful to decide the priority where there are concurrent Write requests. The requester with the higher priority number will get the chance to Write compared to the requester with comparatively a lower priority number. The following level of precedence is followed.

$$PN_1 > PN_2 > \ldots \ldots \ldots PN_i$$

where, ‘$i$’ is the total number of HNs.
Write Operation Priority (WOP): This is a variable to hold the PN of any Head Node, which is currently performing the Write operation on the replica(s). There may chances that the same Head Node requests more than one time for a Write operation. During such requests, if the priority numbers of the earlier requests and the successive requests are compared, the priority numbers will be same which may lead to deadlock due to circular wait. Therefore WOP is used to avoid the circular Write operation which would avoid deadlock due to circular wait.

5.3.2 Service Read Operation

When a Service Consumer node is in need of a Service information it will execute a Service Read Operation. It could be a Service Information Read from a Home SRL node or a Service Information Replica Read from a Foreign SRL node. For every Read operation in the system, there is a set of messages exchanged in the system which is shown in Figure 5.2. The figure shows the message exchanges with respect to time ‘t’ at time ‘t1’, ‘t2’ and ‘t3’. There are four messages exchanged during the Read operation which are 1) \(r_{l\text{msg}}\) for Read Lock Message, 2) \(w_{msg}\) for Wait Message, 3) \(rl_{ack\text{msg}}\) for Read Lock Acknowledge Message and 4) \(ru_{msg}\) for Read Unlock Message.

![Figure 5.2 Message Flow for Service Read Operation](image)

Figure 5.2 Message Flow for Service Read Operation
The proposed methodology for the Service Read operation in the DUDDI system is given in Figure 5.3. There are three phases in the Read Operation which are discussed as follows.

```plaintext
procedure startup(Gm)
    forall HNv in Gm such that HNv ∈ {HN1,HN2,...HNj}
        reset LL, GL, WOP values of HNv
    endfor
endprocedure

procedure readlock()
    for any LNw under HNv such that LNv ∈ {LNv1, LNv2,...LNvj} and HNv ∈ {HN1,HN2,...HNj}
        create rlmsg and forward(rlmsg, HNv)
    endfor
    for any HNv in Gm receive rlmsg from LNw such that HNv ∈ {HN1,HN2,...HNj} and LNw ∈ {LNv1, LNv2,...LNvj}
        if ((HNv, LL is set) or (HNv, GL is set))
            forall LNv under HNv such that LNv ∈ {LNv} U {LNv1, LNv2,...LNvj-1}
                create wmsg and forward(wmsg, LNv)
            endif
        else
            set HNv, LL
            create rlackmsg with replica copy and forward(rlackmsg, LNw)
        endif
    endfor
endprocedure

procedure readacknowledge()
    for any LNw under HNv receive rlackmsg such that LNw ∈ {LNv1, LNv2,...LNvj} and HNv ∈ {HN1,HN2,...HNj}
        create rumsg and forward(rumsg, HNv)
    endfor
endprocedure

procedure readunlock()
    for any HNv in Gm receive rumsg from LNw such that HNv ∈ {HN1,HN2,...HNj} and LNw ∈ {LNv1, LNv2,...LNvj}
        HNv, LL is reset
    endfor
endprocedure
```

Figure 5.3 Procedures for Service Read Operation
**Phase 1:** The Leaf Node which needs to access the latest updated replica issues a Read Lock message ($rl_{msg}$) to its Head Node. When a Head Node receives the read lock message from one of its Leaf Nodes it has to verify whether any other lock has already been acquired by other nodes are not. Therefore, the Head Node after receiving a $rl_{msg}$, it will take any one of the following two cases according to the locks already provided.

**Case-1:** If either a Global Lock (GL) or a Local Lock (LL) of the corresponding Head Node is set, then the corresponding Leaf Node is replied with a $w_{msg}$ message. This Leaf Node understands that the Head Node is busy with some other operation and may wait for a set period (threshold value can be set) before an another $rl_{msg}$ message is issued by that Leaf Node.

**Case-2:** If both Global Lock and Local Lock is not set, then Local Lock of the corresponding Head Node is set and communicates the $rlack_{msg}$ to the read requested Leaf Node, which carries the acknowledgment as well as the copy of the replica.

If the Head Node for which the LL is already set (case-2) but receives a read lock message again, then it propagates a $w_{msg}$, wait message to that Leaf Node to avoid it from sending unnecessary read lock message requests from the other LNs. This is termed as On Demand Lock Forward (ODLF). This methodology reduces the unnecessary message passes among the Distributed Spanning Trees and thereby the message overload in the DUDDI system is reduced. If a particular Leaf Node could not find its Head Node and if there is no acknowledgement for repeated Read request messages, then the corresponding Leaf Node becomes the Head Node and will initiate the procedure to construct the corresponding Distributed Spanning Tree wherein, all the Leaf Nodes of the unfound Head Node will become the members and this scenario is termed as Reclustering. Whenever any Leaf Node is not able to find its Head Node, reclustering is performed.
**Phase 2:** The Read requested Leaf Node is acknowledged by its Head Node. The Head Node also sends a *reply* message that contains the copy of the latest replica of the requested service information. The Leaf Node after receiving the requested data item will send a Read unlock (*ru*$_{msg}$) message to the corresponding Head Node and disengages it to enable the Head Node to take further requests from other Leaf Nodes.

**Phase 3:** When the Local Lock of a Head Node is set for a Leaf Node and receives the local Read unlock (*ru*$_{msg}$) message from the same Leaf Node, then the Head Node’s Local Lock is reset. Now the Head Node checks whether it has sent any wait message to any of its Read lock requested Leaf Nodes and if so, it will send a null message to those Leaf Nodes to invite them again for a new Read lock operation. Since a consistent and latest versioned replica of the services information are ensured in each Head Node through the proposed methodology, verification again for the version or consistency becomes unnecessary.

**5.3.3 Service Write Operation**

Service Write operation is the process performed by the Service Providers to register a new Service or to update the existing Service information replica, with its new set of Service information, maintained by its corresponding SRL node using three phases of message exchanges. The flow of messages with respect to time, ‘*t*’ is shown in Figure 5.4. The figure shows the message flow at various time intervals ‘*t*$_1$’, ‘*t*$_2$’, and ‘*t*$_3$’. Eight different messages are being exchanged during the Write operation which are 1) *wl*$_{msg}$ for Write Lock Message, 2) *w*$_{msg}$ for Wait Message, 3) *wlack*$_{msg}$ for Write Lock Acknowledge Message, 4) *doup*$_{msg}$ for Do Update Message, 5) *up*$_{msg}$ for Update Message, 6) *upack*$_{msg}$ for Update Acknowledge
Message, 7) $commit_{msg}$ for Commit Message and 8) $wu_{msg}$ for Write Unlock Message.

Figure 5.4 Message Flow for Service Write Operation

**Phase 1:** The Leaf Node which needs to perform the Write operation, which could be publish of new services information or could be an update of existing services information or its replica, sends the Write lock message ($wl_{msg}$) to its Head Node. On receiving the Write lock message, the Head Node would perform any one of the following operations. The Phase 1 operations are given in Figure 5.5.

**Case-1:** When a Head Node receives Write lock message from any of its Leaf Nodes, then the Head Node will check its Global Lock (GL) before initiating the following sequence of actions.
Procedure writelock()

for any LN_w under HN_v such that LN_w ∈ {LN_v1, LN_v2...LN_vj} and HN_v ∈ {HN_1, HN_2...HN_i}
create wmsg and forward(rmsg, HN_v)
start writelock timeout
endfor

for any HN_i in G_m receive wmsg from LN_w such that HN_v ∈ {HN_1, HN_2...HN_i} and LN_w ∈ {LN_v1, LN_v2...LN_vj}
if (HN_v.GL is set)
forall LN_x under HN_v such that LN_x ∈ {LN_w}  U {LN_v1, LN_v2...LN_vj-1}
create wmsg and forward(wmsg, LN_x) // ODLF technique
endif
elseif (HN_v(LL is set)
HN_v will wait till it completes its current read operation
HN_v, LL is set
HN_v, GL is set
set HN_v.WOP as HN_v.PN
forall HN_w in G_m such that HN_v ∈ {HN_1, HN_2...HN_i} - HN_v
forward(wmsg, HN_w)
endif
endfor
endfor

for any HN_v in G_m receive wmsg from HN_w such that HN_w ∈ {HN_1, HN_2...HN_i} - HN_v
if (HN_v, GL is set)
if(HN_v.WOP > HN_w.PN)
create wmsg and forward(wmsg, HN_w)
elseif (HN_v, LL is set)
HN_v will wait till it completes its current read operation
HN_v, LL is set
HN_v, GL is set
set HN_v, WOP as HN_v.PN
create wbackmsg and forward(wbackmsg, HN_w)
endif
endif
endfor
end procedure

Figure 5.5 Procedures for Service Update Operation – Phase 1
**Step-1a:** If the Global Lock of the corresponding Head Node is already set (i.e. the Head Node is already engaged in a Write operation), then it will send a wait ($w_{msg}$) message to the corresponding Leaf Node which issued the request for Write lock. The Write request will be initiated again after a while.

**Step-1b:** If the Local Lock of the corresponding Head Node is already set, then it has to wait till the completion of the local read operation before proceeding to the Step-1c.

**Step-1c:** If Global Lock of the corresponding Head Node is reset, then both the Local Lock and the Global Lock will be set. Then it updates its Write Operation Priority (WOP) value as its own Priority Number (PN) and broadcasts the Write lock message to all other Head Nodes in the system.

**Case 2:** When a Head Node ‘$i$’ receives a Write lock message from another Head Node ‘$j$’, then depends on the value of its Global Lock, the following sequence of actions are initiated.

**Step-2a:** If the Global Lock of the Head Node ‘$i$’ is already set, then it checks whether the WOP$_i$ is greater than the PN$_j$ of the Global Lock requesting Head Node ‘$j$’ or not. If WOP$_i$ > PN$_j$, where, WOP$_i$, is the Write Operation Priority value of the Head Node ‘$i$’ for which the Global Lock has already been set and the PN$_j$ is the Priority Number of the Head Node ‘$j$’ which is requesting the Global Lock currently, then the HN ‘$i$’ will reply a wait message to the HN ‘$j$’ and if the WOP$_i$ is not greater than PN$_j$, then Step-2b is initiated.

**Step-2b:** If the Local Lock of the corresponding Head Node is already set, then it waits till the completion of the local Read operation and then proceeds with Step-2c.

**Step-2c:** If Global Lock is not set, then the Global Lock is set and updates its WOP value as the PN$_j$ of the Head Node which requests the Write lock. After the Global Lock is set and WOP is updated, the Head Node ‘$i$’ sends Write acknowledge message to the Head Node ‘$j$’.
During the process of Write operation, if the Head Node is untraceable by a Leaf Node then it aborts the Write request process and initiates the reclustering operation. Also, if a Head Node is not able to find any one of the other Head Nodes during the process of its Write operation, then it induces any of one the Leaf Nodes of that faulty Head Node to proceed with reclustering operation and the details of the Leaf Nodes of the faulty Head Node is found from the details it has from the earlier communications.

**Phase 2:** The Head Node, which requested the Write lock will continue to wait for its Write acknowledge message ($wlack_{msg}$) from all the other Head Nodes. In the mean time, if the Head Node receives any Write lock request with higher WOP, it will cease to wait for the acknowledge message. On receiving each ACK message, the Head Node will check the condition whether $n (wlack_{msg}) = m – 1$ or not. Here, $n (wlack_{msg})$ is the total number of Write acknowledge messages received by the corresponding Head Node and ‘$m$’ is the total number of Head Nodes in the system. If the above condition is TRUE, then the Head Node has to send do update message ($doUp_{msg}$) to the Leaf Node which has requested the Write lock to proceed with Write operation. If it is FALSE, the corresponding Head Node has to wait until it becomes TRUE. The entire operations under Phase 2 of the Service Update operations have been shown in the Figure 5.6.

On receiving the doUpdate message, the Leaf Node which has requested the Write lock creates an update message ($up_{msg}$) with the data to be updated and communicates to its Head Node. After receiving the communication from its Leaf Node, the Head Node updates its replica of the data item and forwards the update message ($up_{msg}$) to all the other Head Nodes of the system. On receiving the update message ($up_{msg}$), every Head Node updates its own replica. After the successful update, each Head Node will send an update acknowledge message ($upack_{msg}$) to the corresponding Head Node which is the origin of the update process.
procedure writeacknowledge()

for any \( H_N \), in \( G_m \) receive \( \text{wlack}_m \) from \( H_N_w \) such that \( H_N_w \in \{H_N_1,H_N_2,...H_N_i\} - H_N_v \)
if \( n(\text{wlack}_m) > i-1 \)
  for any \( L_N_w \), under \( H_N_v \), from which \( \text{wlack}_m \) is received such that \( L_N_w \in \{L_N_1,L_N_2,...L_N_j\} \)
    create \( \text{doup}_m \) and forward(\( \text{doup}_m \), \( L_N_w \))
endfor
else
  \( n(\text{wlack}_m) + 1 \)
endif
endfor

for any \( L_N_w \), under \( H_N_v \), receive \( \text{wlack} \) such that \( L_N_w \in \{L_N_1,L_N_2,...L_N_j\} \) and \( H_N_v \in \{H_N_1,H_N_2,...H_N_i\} \)
  create \( \text{up}_m \) with data to update replica copy replica copy and forward(\( \text{up}_m \), \( H_N_v \))
endfor

for any \( H_N \), in \( G_m \) receive \( \text{up}_m \) such that \( H_N_v \in \{H_N_1,H_N_2,...H_N_i\} \) and \( L_N_w \in \{L_N_1,L_N_2,...L_N_j\} \)
if \( H_N_v \) received \( \text{up}_m \) from \( L_N_w \)
  \( H_N_v \) updates its replica with data from \( \text{up}_m \)
  forall \( H_N_u \), in \( G_m \), such that \( H_N_u \in \{H_N_1,H_N_2,...H_N_i\} \) - \( H_N_v \)
    forward(\( \text{up}_m \), \( H_N_u \))
endfor
elseif \( H_N_v \) received \( \text{up}_m \) from \( H_N_w \)
  \( H_N_v \) updates its replica with data from \( \text{up}_m \)
  create \( \text{upack}_m \) and forward(\( \text{upack}_m \), \( H_N_v \))
endif
endfor

for any \( H_N \), in \( G_m \) receive \( \text{uplack}_m \) from \( H_N_w \) such that \( H_N_w \in \{H_N_1,H_N_2,...H_N_i\} \) - \( H_N_v \)
if \( n(\text{upack}_m) > i-1 \)
  for any \( L_N_w \), under \( H_N_v \), from which \( \text{up}_m \) is received such that \( L_N_w \in \{L_N_1,L_N_2,...L_N_j\} \)
    create \( \text{commit}_m \) and forward(\( \text{commit}_m \), \( L_N_w \))
endfor
else
  \( n(\text{upack}_m) + 1 \)
endif
endfor
endprocedure

Figure 5.6 Procedures for Service Update Operation – Phase 2

124
After receiving the update acknowledge message \((\text{upack}_\text{msg})\) from all the other Head Nodes of the system, the Head Node which was requested the Global Lock will propagate a commit message \((\text{commit}_\text{msg})\) to the Leaf Node which acquired the Global Lock and proceeds to Phase 3. In this entire process, the unacknowledged messages are to sent back to avoid inconsistent updates, so that every Head Node in the DUDDI system will have a globally consistent replica.

**Phase 3:** The Head Node on which the Write operation is completed will reset its Global Lock, Local Lock and WOP values and forward the Write unlock message \((\text{wu}_\text{msg})\) to all the other Head Nodes as shown in Figure 5.7. Every Head Node, upon receiving this message performs the same operation whereby the Global Lock, Local Lock and WOP of all of them will be reset.

```
procedure writeunlock()
    for any \(HN_v\) in \(G_m\) completed phase 1 and 2 such that \(HN_v \in \{HN_1, HN_2...HN_i\}\)
        \(HN_v\).LL is reset
        \(HN_v\).GL is reset
        \(HN_v\).WOP is reset
    endfor

    forall \(HN_w\) in \(G_m\) such that \(HN_w \in \{HN_1, HN_2...HN_i\} - HN_v\)
        create \(\text{wu}_\text{msg}\) and forward(\(\text{wu}_\text{msg}, HN_v\))
    endfor

endprocedure
```

Figure 5.7 Procedures for Service Update Operation – Phase 3
5.4 SUMMARY

In the Chapter 4 a large scale DST Structured P2P based DUDDI system has been modeled and simulated in OMNET++. The simulated system has been used here to study the proposed methodologies for an efficient node search (Section 5.2) in the DST structured P2P DUDDI. The Proposed Node search methodologies use the details stored by every Head Node regarding its Leaf Nodes and this approach makes the node/resource search operation in a large scale P2P based DUDDI system more efficient and economic with very minimum number of message passes and reduced routing table entries in every node of the system. The simulated system has also been utilized as a test bed to study the proposed methodologies for the (Section 5.3) Service Register (a Write operation), Service Update (Write operation) and Service Read (Read operation). The proposed methodologies ensured the consistency of the Service Replica at all Head Nodes wherever the Services Information have been replicated. This chapter also demonstrates the way how the Replica Management is done using Distributed Spanning Trees, which effectively considered the message pass, message bottlenecks and replica consistency. The effectiveness of the proposed approach has been quantified and analyzed in Chapter 6.