

1 PAXOS-Implementierung (Aufg. 6 - Muster)

PaxosNode: Node	PaxosStarter: Thread
<code>propose(value): round</code> <code>getDecision(round): value</code>	<code>run</code> <code>executeEvent</code>
<code>setLeader</code> <code>restart</code>	<code>startround</code> <code>finishedround</code> <code>receive</code>
<code>receive</code> <code>rx_collect, rx_last, rx_prepare,</code> <code>rx_accept, rx_success, phase1,</code> <code>phase2, phase3</code>	<code>roundFailed</code> <code>nodeFailed</code> <code>nodeOK</code> <code>getLeaderFromSet</code>
<code>getNewUniqueSeqno</code> <code>majority</code> <code>getNextEmptyRound</code> <code>getMaxKnownRound</code>	

1 PAXOS-Implementierung

■ PaxosNode: Zustandsvariablen

- NodeID leader
- int maxKnownSeqNum

- RoundHash proposed
 Round r => Wert v mit propose vorgeschlagen
- RoundHash prepared
 Round r => (SeqNo s, Wert v) über Begin erhalten
- RoundHash decided
 Round r => Wert v über Success erhalten

- Hashtable recv_last
 Für jeden Knoten: Last-Antwort (aktuelle SeqNum)
- RoundHash recv_accept
 Round r => empfangene Accept (aktuelle SeqNum)

1 PAXOS-Implementierung

■ Start des Algorithmus (Phase 1)

```
◆ setLeader(NodeID newLeader) {
    this.leader = newLeader;
    if(newLeader.equals(this.getID())) phase1();
}
◆ restart() { phase1(); }
◆ phase1() {
    maxKnownSeqNum = getNewUniqueSeqNum();
    max = getMaxKnownRound();
    recv_last = new Hashtable(); recv_accept=new RoundHash();
    PaxosMessage msg=new PaxosMessage.createCollect(
        maxKnownSeqNum, max);
    sendall(msg);
    starter.startround(-1);
}
```

1 PAXOS-Implementierung

■ Antwort auf Collect

```
◆ rx_collect(NodeID sender, int seqno, int knownround) {
    if( seqno<maxKnownSeqNum) { sendOldRound(...); return; }
    maxKnownSeqNum = seqno;
    RoundHash last_list = prepared.getAbove(knownround);
    last_list.putAll( decided.getAbove(knownround);
    PaxosMessage msg = PaxosMessage.createLast(seqno, last_list);
    send(sender, msg);
}
```

1 PAXOS-Implementierung

■ Phase 2 des Algorithmus

```
◆ rx_last(NodeID sender, int seqno, Hashtable last_list) {
    if( seqno!=maxKnownSequenceNumber) { sendOldRound(...);return;}
    if( !this.getID().equals(leader) ) return;
    if( majority(recv_last) ) return; // Phase 2 bereits ausgeführt
    recv_last.put( sender, last_list );
    if( majority(recv_last) ) {
        starter.finishedround(-1)
        Alle Knoten i, alle Runden r:
            recv_last{i}->last_list{r}: DECIDED(v) -> phase3(r, v)
        Alle Knoten i, alle Runden r:
            recv_last{i}->last_list{r}: PREPARED(seqno, v)
            -> Suche zu r den Wert v mit maximaler seqno
        Für alle Runden r:
            Falls solcher Wert v gefunden: phase2(r, v)
            Ansonsten, falls proposed(r):= v' existiert: phase2(r, v')
    }
}
```

1 PAXOS-Implementierung

```
◆ phase2(int round, String value) {
    starter.startround(round);
    PaxosMessage msg = PaxosMessage.createPrepare(round,
        maxKnownSeqNum, value);
    sendall(msg);
}
◆ rx_prepare(NodeID sender, int round, int seqno, String value) {
    if(seqno != maxKnownSeqNum) return;
    prepared.put(round, [ maxKnownSeqNum, value ] );
    PaxosMessage msg = PaxosMessage.createAccept(round, seqno);
    send(sender, msg);
}
◆ rx_accept(NodeID sender, int round, int seqno) {
    if(seqno != maxKnownSeqNum && not leader) return;
    if( majority(recv_accept) ) return; else recv_accept.add(sender);
    if( majority(recv_accept) ) phase3(round, prepared.get(round)[1] );
}
```

1 PAXOS-Implementierung

■ Phase 3 des Algorithmus

```
◆ phase3(int round, String value) {  
    starter.finishedround(round);  
    PaxosMessage msg = PaxosMessage.createSuccess(round, value);  
    sendall(msg);  
}
```

1 PAXOS-Implementierung

■ Hilfsfunktionen

```
◆ private int max_known = -1;  
  public int getMaxKnownRound() {  
    while( decided.get(max_known+1) != null ) max_known++;  
    return max_known;  
  }  
◆ private int max_empty = 0;  
  private int getNextEmptyRound() {  
    while( proposed.get(max_empty) != null || prepared.get(max_empty) !=  
          null || decided.get(max_empty) != null ) max_empty++;  
    return max_empty;  
  }  
◆ private int getNewUniqueSeqno() { // Annahme: maximal 256 Knoten  
  seqno = (maxKnownSeqNum & 0x7ffff00) + 0x100 + this.getID().num();  
  return seqno;  
}
```

1 PAXOS-Implementierung

■ PaxosStarter

```
◆ public void run() {
    execute("PING"); // Sendet PING und trägt PING-Event in events ein
    while(true) {
        synchronized(this) {
            long timeout = events.getNextTime();
            long now = System.currentTimeMillis();
            try {
                wait(timeout-now);
            } catch(InterruptedException ex) {}

            now=System.currentTimeMillis();
            String str;
            while( (str = events.getEvent(now)) != null) {
                execute(str);
            }
        }
    }
}
```

1 PAXOS-Implementierung

```
◆ execute(String event) {
    if(event.startsWith("PING")) {
        node.sendall(PaxosMessage.createPing());
        events.put("PING", new Long(now+PING_TIME));
    } else if (event.startsWith("NODE")) {
        nodeFailed(event);
    } else if (event.startsWith("ROUND")) {
        node.restart();
    }
}

◆ receive(Message msg) {
    nodeOk(msg.sender());
}
```

nodeOK / nodeFailed rufen ggf. node.setLeader(leader) auf