Asynchronous Multi-Agent ASMs with real-time constraints

Processor-Group Membership Protocol Specification and Verification

Egon Börger

Dipartimento di Informatica, Universita di Pisa
http://www.di.unipi.it/~boerger

For details see Chapter 6.3 (Time-Constrained Asynchronous ASMs) of:

E. Börger, R. Stärk

Abstract State Machines

A Method for High-Level System Design and Analysis

Springer-Verlag 2003

For update info see AsmBook web page:

http://www.di.unipi.it/AsmBook

Group Membership Problem

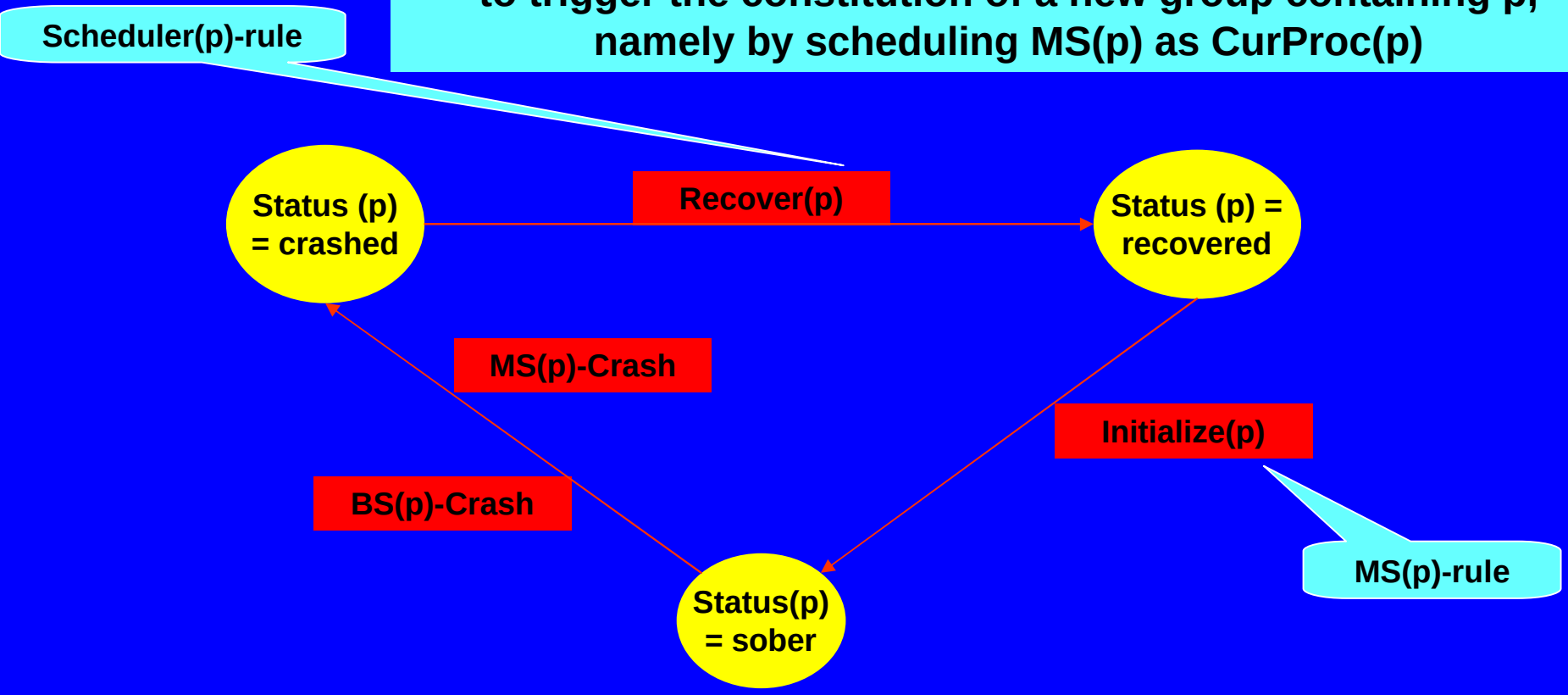
- The Context. Fault tolerance for distributed computing services can be provided by
  - letting members of a server group cooperate to provide the services
  - replicating the relevant service state info at all sites
- Group Membership Protocols ensure that despite information propagation delays and server failures, the service state info stored at each group member
  - remains up-to-date
  - is the same for all group members (in the steady state)
- Special Case Processor-Group Membership: via message exchange a global agreement has to be achieved about the set of all correctly functioning processors in the system

The Processor-Group Members

- PROCESSOR: finite static domain of processors p with
  - Clock(p) : REAL strictly increasing monitored fct
  - Status(p) : {sober, crashed, recovered} controlled fct
  - CurProc(p) the currently running process, e.g. MS(p), BS(p) below, executing tasks from deadline to completion with earliest-deadline-first scheduling
    - Dline(x) = minimum of deadlines of tasks to be handled by x
  - Membership Server MS(p)
    - handling entry of p into a new processor-group after recovery from a crash: Initialize(p), including broadcast of a new-gp msg
    - maintaining processor group info - Group(p) and membership view Members(p) - while p is alive, by handling incoming
      - messages announcing the constitution of a new group
      - messages reporting which processors are present in a group
  - Broadcast Server BS(p) sending with period d_h (heartbeat) broadcast msgs to all processors

Fault tolerance lifecycle of a processor p

scheduling processes on p includes, upon recovery of p, to trigger the constitution of a new group containing p, namely by scheduling MS(p) as CurProc(p)



Two reasons for crashing (processor interrupt):

- **broadcast : BS(p) was scheduled too late** — **Clock(p) > BCastTime(p)**
- **new-group msg because MS(p) was scheduled too late to handle it** — **Clock(p) > Timestamp(CurMsg(p))**

Performance Failures

- A Performance Failure occurs at p if p is sober but has a pending task whose deadline has been exceeded
  - otherwise p is called correct
- It is assumed that performance failures can be detected and turned into crashes

Messages

- MESSAGE = MSGTYPE x REAL x $2^{PROCESSOR}$
- For m = (x,y,z) let

  Type(m) = x, Timestamp(m) = y, View(m) = z

  Deadline(m) =
  - Timestamp(m) if Type(m) = new-gp
  - Timestamp(m) + d_n if Type(m) = present

  new-group msgs are scheduled with delay d_n > maximum network propagation delay so that every correctly functioning processor has a chance to see them
- Timestamp(m) records the time the group was created by Initialize(p). It is used here as group Id.
- View(m) represents the set of all members of processor group Timestamp(m).

Messages (Cont'd)

- CurMsg(p) = the msg currently seen by MS(p). Since CurMsg(p) should always store the msg with earliest deadline, set Dline(MS(p)) = Deadline(CurMsg(p))
- InBox(p) $\subseteq$ MESSAGE : a controlled set containing the msgs delivered to p from the network, via a monitored fct InMsg(p), but not yet seen, as CurMsg(p), by MS(p)
- AptMsg(p)

= relevant InBox(p)-msgs with Deadline $\geq$Clock(p)- d_u

= {m$\in$ InBox(p)|Deadline(m) $\geq$Clock(p)- d_u &
Timestamp(m) $\geq$ StartUpTime(p)}

where scheduling uncertainty d_u : a task with deadline d may not be scheduled until d- d_u

StartUpTime(p)= recovery time of p from last crash

Message Passing System

- Message Passing System modeled by abstract propagation of messages to the network & two intermediary agents:
  - Broadcast m, say by setting InTransit(m):=true for some predicate InTransit over messages, shared by all MsgCarrier(p)
  - MsgCarrier(p) delivers to InBox(p) every message InMsg(p) which is incoming from a Broadcast (i.e. from InTransit)
  - Custodian(p) delivers from the subset AptMsg(p) of InBox(p) a message with the minimal timestamp to MS(p), namely as CurMsg(p)
- Reliability Assumption: every message which has been broadcast by a processor will be delivered to every processor
- Delivery Bound relating different processor clocks: a msg sent (which in this protocol means broadcast) by p at Clock(p)-time t_1 will be received by q at Clock(q)-time t_2 within a carry delay d_c :

$$0 < t_2 - t_1 \leq d_c$$

Scheduling Requirements

- earliest-deadline-first:
  - Scheduler(p) provides for the next process CurProc(p) the enabled process among MS(p) or BS(p) with minimal deadline
  - Custodian(p) provides for CurMsg(p) always a message with the earliest deadline
- non-preemptive:
  - Scheduler(p) runs only when CurProc(p) has been switched from MS(p) or BS(p), after their completion, to undef. This realizes the requirement that only processors can be interrupted, whereas tasks run until their completion.
- scheduling uncertainty: a task with deadline d may not be scheduled until $d\text{-}d_u$ where d_u is the system wide upper uncertainty bound. Therefore processes on p are called enabled only when their deadline is $\geq$Clock(p)- d_u :

EnabledBS(p) iff DlineBS(p) $\geq$Clock(p)- d_u

Dline(BS(p)) = BCastTime(p)

EnabledMS(p) iff CurMsg(p)$\neq$undef & DlineMS(p) $\geq$Clock(p)- d_u

Agents of the processor-group membership protocol

- Five subagents for each processor p
  - Scheduler(p) assigns as next CurProc(p) the enabled process among MS(p) or BS(p) with minimal deadline, including scheduling MS(p) upon crash recovery of p
    - a lower recovery bound d_r is assumed for the time bw a crash and a subsequent recovery
  - Membership Server MS(p)
    - initializes p upon recovery
    - maintains processor group info, reacting to CurMsg(p) by MS(p)-crash, or synchronizing p with new-gp, or updating Members(p)
  - Broadcast Server BS(p) periodically broadcasts presence msgs for p & detects broadcast crashes at p
  - MsgCarrier(p) collects for p msgs incoming from the network, adding them to InBox(p)
  - Custodian(p) delivers a msg with earliest deadline from InBox(p) to CurMsg(p) of MS(p)

6 Goals of the processor-group membership protocol

- Stability of local views. Once a processor joins a group g, it stays in g until either a processor fails or one recovers and attempts to rejoin.
- Agreement on history. If two processors are joined to a common group g and none of them crashes between joining g and joining the next group, then that next group is the same for both processors.
- Agreement on group membership. Alive processors in a group have the same membership view.
- Reflexivity. The membership view of an alive processor in a group contains that processor.
- Bounded join delay. There is a constant join delay d_j s.t. if a processor becomes sober at time t then, by time $t + d_j$, it will join a new group along with every other processor that stays correct in the time interval $[t,t + d_j]$.
- Bounded failure detection delay. There is a constant failure delay d_f s.t. if a processor belonging to a group g fails at time t then, by time $t+ d_f$, all the members of g that stay correct in $[t,t + d_f]$ will join a new group g' not containing p.

Teaming member subagents
for processor-group membership Protocol Runs

- We assume the five subagents for each processor p to be teamed as a basic ASM M(p)
  - e.g. choosing non-deterministically for each move of M(p) one enabled subagent to fire, defining M(p) as

    choose R∈{Scheduler(p), MS(p), BS(p), MsgCarrier(p), Custodian(p)} in

    R
- The choice is bw one of Scheduler, MS, BS – which have disjoint guards - and one of MsgCarrier, Custodian.
  - This avoids possibly conflicting updates of CurMsg (by MS and Custodian) and of InBox (by MsgCarrier and Custodian)

Protocol Runs for processor-group membership

- The monitored fcts of M(p) are Clock(p), InMsg(p), all other dynamic fcts are controlled or derived
- Initial states: Status(p)=crashed, InBox(p)=∅, all of CurMsg(p), Group(p), Members(p), StartUpTime(p), BCastTime(p) have value undef
- Protocol Runs are defined as multi-agent asynchronous runs of the basic machines M(p), $p \in$ PROCESSOR, which (are assumed to) respect
  - the lower Recovery Bound d_r on recovery time
  - the upper Delivery Bound d_c on message delivery time over the network
    - These constraints on runs abstract from explicit lower/upper time bounds one could incorporate into the related rules of M(p)
- NB. Passing of time can be taken as reflecting not furthermore specified moves of other processes

Scheduler(p) rules assigning values to CurProc(p)

- **Recover(p)** ≡ upon recovery of p, MS(p) has to initialize p

  If Status(p) = crashed then Status(p) := recovered
  CurProc(p) := MS(p)

- **Schedule(p)** ≡ scheduling protocol processes MS(p), BS(p)

  If CurProc(p) = undef & Status(p) = sober non-preemptiveness
  & (AptMsg(p) = ∅ or CurMsg(p) ≠ undef)
  then CurProc(p) := MinEnabled(p) earliest-deadline-first
  where MinEnabled(p) =
  x if (Enabledx & not Enabledy)
  or (Enabledx & Enabledy & Dline(x)<Dline(y))
  y if (Enabledx & Enabledy & Dline(x) ≥ Dline(y))
  for (x,y) ∈{(MS(p),BS(p)), (BS(p),MS(p))}

NB. CurMsg(p) is defined when via the Custodian(p)-rule below it has been set to a msg from InBox(p) with minimal deadline

MS(p)-Initial rule to initialize recovered processors

- MS(p)-Initialize≡

  If Status(p) = recovered & CurProc(p) = MS(p)

  then Reset(p)

  ResetGroupInfo(p)

  StartUpTime(p) := Clock(p) + d_n

  BroadcastNewGroup(p)

  Status(p) := sober

  CurProc done

- Reset(p) ≡ BCastTime(p) := undef, CurMsg(p) := undef
- ResetGroupInfo(p) ≡ Group(p) := undef, Members(p) := undef
- BroadcastNewGroup(p) ≡

  Broadcast (new-gp, Clock(p) + d_n,{p})
- CurProc done ≡ CurProc(p) := undef

MS(p)-Crash rule
handling miss of a new-gp msg

- MS(p)-Crash ≡

  p crashes upon deadline miss of a new-gp msg

If Status(p) = sober & CurProc(p) = MS(p) &
Type(CurMsg(p))= new-gp

then If Clock(p) > Timestamp(CurMsg(p))
then Status(p) := crashed

MS(p)-NewGroup rule for timely handling of new-gp msg

MS(p)-NewGroup ≡

If Status(p) = sober & CurProc(p) = MS(p) &
Type(CurMsg(p))= new-gp

then If Clock(p) ≤ Timestamp(CurMsg(p)) then

BroadcastRegistration(p)

SynchronizeWithNewGp(p)

CurMsg done

CurProc done

BroadcastRegistration(p) ≡ reporting p as present in new-gp

Broadcast (present, Timestamp(CurMsg(p)) ,{p})

SynchronizeWithNewGp(p) ≡

BCastTime(p) := Timestamp(CurMsg(p)) + d_h

synchronize heartbeat of p to new-gp canceling any pending heartbeat

CurMsg done ≡ CurMsg(p) := undef

MS(p)-ChangeGroup rule handling arrival of presence reports

MS(p)-ChangeGroup ≡ If Status(p) = sober &
CurProc(p) = MS(p) & type(CurMsg(p))= present
then CurMsg done
CurProc done
If Members(p) ≠ View(CurMsg(p))
then UpdateGroupMembership(p)

View(m) yields the processors reported as present in m's group
NB.Groups have as identifiers the timestamp of their creation
here Timestamp(m)

No group change if p's group has not changed

UpdateGroupMembership(p) ≡ change to new group
Members(p) := View(CurMsg(p))
Group(p) := Timestamp(CurMsg(p))

BS(p)-Crash rule
handling miss of a heartbeat

- BS(p)-Crash ≡

  p crashes upon deadline miss of a broadcast

If Status(p) = sober & CurProc(p) = BS(p)

then If Clock(p) > BCastTime(p)

then Status(p) := crashed

BS(p)-Presence rule for timely handling of heartbeats

BS(p)-Presence ≡ within broadcast time, p reports its presence & updates its next heartbeat deadline

If Status(p) = sober & CurProc(p) = BS(p)

then If Clock(p) ≤ BCastTime(p) then

BroadcastPresence(p)

BCastTime(p) := BCastTime(p) + d_h

CurProc done

BroadcastPresence(p) ≡ Broadcast (present, BCastTime(p), {p})

reporting p as present in its group BCastTime(p)

MsgCarrier(p) rule transfering incoming msgs to InBox(p)

MsgCarrier(p) ≡

If InMsg(p) ≠ undef then

Let (a,b,c) = InMsg(p) in

If a = new-gp then — add new-gp msg

InBox(p):= InBox(p) ∪{InMsg(p)}

elseif InBox(p) has no msg with Timestamp b then

InBox(p):= InBox(p) ∪{InMsg(p)}

add presence msg from newly constituted group

else let m = ιx(x ∈ InBox(p): Timestamp(x)=b) in

InBox(p):= (InBox(p)–{m}) ∪{(a,b,View(m) ∪{c})}

bundle msgs into one per group (i.e. with same timestamp)

Custodian(p) rule assigning CurMsg(p) to MS(p)

Custodian(p) ≡

If Status(p) = sober & CurMsg(p) = undef &

AptMsg(p) contains a msg with minimal deadline

then deliver m from InBox(p) to MS(p)

where m = ε x (x has minimal deadline in AptMsg(p))

deliver m from InBox(p) to MS(p) ≡

InBox(p):= InBox(p)–{m}

CurMsg(p):= m

NB. This rule realizes the protocol requirement that the membership servers see the msgs in their deadline order

Notations for States of Protocol Runs

- $\Sigma(p)$: the signature of M(p) without InTransit
- $\Sigma^{-}(p)$ = $\Sigma(p)$ without monitored Clock(p), InMsg(p)
- For states S of p (really of $\Sigma(p)$) let S^{-} be its restr to $\Sigma^{-}(p)$
- S_n : the n-th state of p, t_n : the value of t in state S_n
- d_r-recovery bound & d_c-delivery bound constraints on protocol runs can be made precise as follows, for all p,q :
  - If p gets crashed in S_n and recovered in S_{n+k} for some k>0, then $Clock(p)_{n+k} - Clock(p)_n \geq d_r$
  - If p sends msg=(type,time,view) to q (i.e. sets InTransit(msg) to true) in state S_n of p, then there is a unique state S_k of q such that
    - $InMsg(q)_k$ = msg
    - $InBox(q)_{k+1}$ contains some (type,time,view') with view' $\supseteq$ view
    - $0 < Clock(q)_k - Clock(p)_n \leq d_c$

Correctness of processors in Protocol Runs

- Correct(p) in state S_n iff no pending task has exceeded deadline:
  - $Clock(p)_n \geq StartUpTime(p)_n$
  - $\forall\, m \in InBox(q)_n$: if $Timestamp(m) \geq StartUpTime(p)_n$ then $Deadline(m) \geq Clock(p)_n$
  - $BCastTime(p)_n \geq Clock(p)_n$
- Correct(p) in real-time interval [t,t'] iff Correct(p) in every state S_n with $k \leq n \leq k'$ where $Clock(p)_k \leq t < Clock(p)_{k+1}$ & $Clock(p)_{k'-1} < t' \leq Clock(p)_{k'}$
- Following these lines to make intuitive protocol related notions rigorous, the above mentioned six goals of the group membership protocols can be turned into precise statements on protocol runs, becoming theorems which have been proved in [GurevichMani1995].

Exercises to the Processor-Group Membership Protocol

- Define the <-relation for moves of agents M(p) in (asynchronous multi-agent) protocol runs.
  - Hint: use the broadcast reliability assumption.
- Prove the finite history property for protocol runs.
  - Hint: use the finiteness of PROCESSOR, the monotonicity of every Clock(p), the positive heartbeat interval $d_h > d_u$, the lower recovery bound $d_r > d_h + d_u$, the new-gp timestamp increment $d_n > d_c + d_u$, and the upper carry-over bound d_c .
- Prove the coherence property for protocol runs.
- Modify the related rules of M(p) to include the lower Recovery Bound d_r on recovery time and the upper Delivery Bound d_c on message delivery time over the network, so that the two constraints on these bounds can be deleted from the notion of protocol runs.

Exercises to the Processor-Group Membership Protocol

- Modify Broadcast m to keep the size of InTransit bounded.
  - Hint: Instead of sending a single copy of m to the entire group, make InTransit binary to send one copy to each processor p, to be deleted by the MsgCarrier(p) when put into InBox(p).
- Modify M(p) to keep the number of messages in InBox(p) bounded, deleting unselectable messages, e.g. those arriving while p is crashed so that they may never be seen by MS(p) and never be removed from InBox(p).
  - Hint: Modify the Custodian(p)-rule to delete messages
    - with timestamp < Clock(p) if p is crashed
    - with timestamp < StartUpTime(p) if p is alive

References

- F. Cristian: Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems.
  - In: Distributed Computing, 6: 175-187, April 1991
- Y. Gurevich and R. Mani: Group Membership Protocol: Specification and Verification
  - In: Specification and Validation Methods (Ed. E. Börger). Oxford University Press, 1995, 295-328
- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis Springer-Verlag 2003, see http://www.di.unipi.it/AsmBook