MODULE *optishopylist*

EXTENDS TLC, $Integers$, $FiniteSets$, $Sequences$, $Reals$

CONSTANTS $PRODUCTS$, $APPS$, IDs, $GATEAPPS$
ASSUME $Cardinality(APPS) > 0$

$PT \triangleq$ INSTANCE PT

$set ++ item \triangleq set \cup \{item\}$
$set -- item \triangleq set \setminus \{item\}$

In shoppying list, the product is in fact the identifier. Any item could have an information for how much of a product one wants to buy (not relevant in this specification).

$ShopyItems \triangleq [id : PRODUCTS,\ bought : \text{BOOLEAN}\ ]$

$ADD\_ACTION \triangleq$ "add"
$RM\_ACTION \triangleq$ "rm"
$SET\_BOUGHT\_ACTION \triangleq$ "set_bought"
$REQ\_SYNC\_ACTION \triangleq$ "req_sync"
$RESP\_SYNC\_ACTION \triangleq$ "resp_sync"
$END\_SYNC\_ACTION \triangleq$ "end_sync"

Actions is the set of all possible actions in the system.

$Actions \triangleq \{$
$\quad ADD\_ACTION,$
$\quad RM\_ACTION,$
$\quad SET\_BOUGHT\_ACTION,$
$\quad REQ\_SYNC\_ACTION,$
$\quad RESP\_SYNC\_ACTION,$
$\quad END\_SYNC\_ACTION$
$\}$

$SyncActions \triangleq \{REQ\_SYNC\_ACTION,\ RESP\_SYNC\_ACTION\}$

SyncMsgs is the set of all possible messages sent for synchronisation of shopy lists.

$SyncMsgs \triangleq$
$\quad [id : IDs,$
$\quad\ app : APPS,$
$\quad\ list : \text{SUBSET}\ ShopyItems,$
$\quad\ mergedList : \text{SUBSET}\ ShopyItems,$
$\quad\ type : SyncActions]$

Messages sent for joining the network.

$JoinRespMsgs \triangleq$
$\quad [app : APPS,$
$\quad\ knownHosts : PT!SeqOf(APPS,\ Cardinality(APPS))]$

$JoinReqMsgs \triangleq$
$[app : APPS]$

Messages sent for notifications about new joiners in the network.

$JoinNotifMsgs \triangleq$
$[app : APPS]$

The spec now depicts a shopping-list *app* where the server *app* manages several users and hence multiple lists of items that synch eventually.

The list contains unique items, thus we use a set.

--algorithm *OptiShopyList*

variable

whether an *app* is a gate
$isGate = [a \in APPS \mapsto a \in GATEAPPS]$,
one shopping list for all *APPS*
$shopyList = [a \in APPS \mapsto \{\}]$,
sync *shopyList* requests/responses
$syncReqQueue = [a \in APPS \mapsto \langle\rangle]$,
$syncRespQueue = [a \in APPS \mapsto \langle\rangle]$,
join to network requests/responses
$joinReqQueue = [a \in APPS \mapsto \langle\rangle]$,
$joinRespQueue = [a \in APPS \mapsto \langle\rangle]$,
new joiner notifications
$newJoinerNotif = [a \in APPS \mapsto \langle\rangle]$,
set of taken *IDs*
$takenIDs = \{\}$ **;**

define

A couple of helpers for shopy-list items

$NewShopyItem(list) \triangleq$
$[id \mapsto (\text{CHOOSE } x \in PRODUCTS : \neg\exists\, i \in list : x = i.id),$
$bought \mapsto \text{FALSE}]$

$ExistingShopyItem(list) \triangleq \text{CHOOSE } x \in list : \text{TRUE}$

$ExistingNotBoughtShopyItem(list) \triangleq \text{CHOOSE } x \in list : x.bought = \text{FALSE}$

Helpers for *Sync* messages request/response.

$NewSyncMsg(id,\ a,\ l,\ ml,\ t) \triangleq$
$[id \mapsto id,$
$app \mapsto a,$
$list \mapsto l,$
$mergedList \mapsto ml,$

$type \mapsto t]$

$NewSyncReqMsg(a,\ l,\ ml,\ t) \triangleq$
$NewSyncMsg($
$(\text{CHOOSE } i \in IDs : \forall\, ti \in takenIDs : i \neq ti),$
$a,\ l,\ ml,\ t$
$)$

$NewSyncReq(app) \triangleq$
$NewSyncReqMsg(app,\ shopyList[app],\ \{\},\ REQ\_SYNC\_ACTION)$

$NewSyncResp(app,\ mergeResult,\ id) \triangleq$
$NewSyncMsg(id,\ app,\ shopyList[app],\ mergeResult,\ RESP\_SYNC\_ACTION)$

Helpers for the decentralized network features.

$NewJoinReqMsg(app) \triangleq [app \mapsto app]$

$NewJoinRespMsg(app,\ hosts) \triangleq$
$[app \mapsto app,$
$knownHosts \mapsto hosts]$

$GateApps \triangleq \{a \in APPS : isGate[a]\}$

$NewJoinerNotifReq(app) \triangleq [app \mapsto app]$

Amongst 'knownApps', wisely choose an element different than 'app'.

$PickGossipFriends(app,\ knownApps) \triangleq$
LET
$Opposit \triangleq$
LET $I \triangleq PT!Index(knownApps,\ app)$IN
$I - (Len(knownApps) \div 2) +$
$(\text{IF } I < Len(knownApps) \div 2 \text{ THEN } Len(knownApps) \text{ ELSE } 0)$

$KnownApps(exclude) \triangleq$
$PT!ReduceSeq($
LAMBDA $a,\ acc : \text{IF } a \in exclude \text{ THEN } acc \text{ ELSE } Append(acc,\ a),$
$knownApps,\ \langle\rangle)$

$Between(val,\ min,\ max) \triangleq$
$PT!Max(min,\ PT!Min(max,\ val))$

$FirstGossip \triangleq \text{LET } ka \triangleq KnownApps(\{app\})\text{IN}$
$\{ka[Between(Opposit - 1,\ 1,\ Len(ka))]\}$

$SecondGossip \triangleq \text{LET } ka \triangleq KnownApps(\{app\} \cup FirstGossip)\text{IN}$
IF $Len(ka) > 0$
THEN $\{ka[Between(Opposit,\ 1,\ Len(ka))]\}$

```
        ELSE {}
    IN  FirstGossip ∪ SecondGossip
```

Merge two lists of *apps* together without duplicating equal elements.

```
MergeKnownApps(apps1, apps2) ≜
    LET AppSeq(n) ≜ PT!SeqOf(APPS, n)

        Contains(appSeq, appItem) ≜
            Cardinality(PT!Matching(appSeq, appItem)) > 0

        f[args ∈ AppSeq(Len(apps1))
                 × AppSeq(Len(apps2))
                 × AppSeq(Len(apps1) + Len(apps2))] ≜

            LET l1 ≜ args[1]

                l2 ≜ args[2]

                acc ≜ args[3]

                PickFromL1 ≜ f[⟨
                    Tail(l1),
                    l2,
                    Append(acc, Head(l1))⟩]

                SkipOneL1 ≜ f[⟨
                    Tail(l1),
                    l2,
                    acc⟩]

                PickFromL2 ≜ f[⟨
                    l1,
                    Tail(l2),
                    Append(acc, Head(l2))⟩]

                SkipOneL2 ≜ f[⟨
                    l1,
                    Tail(l2),
                    acc⟩]
            IN
                IF Len(l2) = 0
                THEN IF Len(l1) = 0
                        THEN acc
                        ELSE IF Contains(acc, Head(l1)) THEN SkipOneL1 ELSE PickFromL1
                ELSE IF Head(l1) ≠ Head(l2)
                        THEN IF Contains(acc, Head(l2)) THEN SkipOneL2 ELSE PickFromL2
                        ELSE IF Contains(acc, Head(l1)) THEN SkipOneL1 ELSE PickFromL1
```

```
        IN   f[⟨apps1, apps2, ⟨⟩⟩]
end define ;

macro notify(apps, newJoiner)
begin
    newJoinerNotif := [a ∈ APPS ↦
        IF a ∈ apps
         THEN Append(newJoinerNotif[a], NewJoinerNotifReq(newJoiner))
         ELSE newJoinerNotif[a]] ;
end macro ;
```

This proccess represents a shopy-list running in one of the several network clients.

Since Opti-shopylist is a decentralized program, the user creates a network of connected instances of Opti-shopylist.

We assume that every client *app* has only one shopy-list.

```
fair + process ClientApp ∈ APPS
variables
```

On network outage, an *app* can't pull new messages.

```
    networkOutage = FALSE,
    joined = FALSE,
    gossipFriends = {},
    knownApps = ⟨self⟩ ;

begin AppLoop:
    while TRUE do
```

MANAGE DECENTRALIZED NETWORK

```
        either
```

SEND JOIN REQUEST

A client *app* might send a join request to any available gate *app*.

```
            await ¬joined ;
            with a ∈ (GateApps −− self)
             do
                joinReqQueue[a] := Append(joinReqQueue[a], NewJoinReqMsg(self)) ;
            end with ;
        or
```

NETWORK *OUTAGE*

```
            networkOutage := TRUE ;
        or
```

RESPOND TO JOIN REQUEST

Any gate *app* receiving a join request will :
- respond with currently known joined *apps*,
- pick new gossip friends,
- *notify* its gossip friends.

```
    if isGate[self] then
        await ¬networkOutage ;
        await joinReqQueue[self] ≠ ⟨⟩ ;
        with joinRequest = Head(joinReqQueue[self]),
             updatedKnownApps = Append(knownApps, joinRequest.app)
         do
            PULL FROM REQ QUEUE
            joinReqQueue[self] := Tail(joinReqQueue[self]) ;

            RESPOND TO REQUESTER
            joinRespQueue[joinRequest.app] := Append(
                joinRespQueue[joinRequest.app],
                NewJoinRespMsg(self, SelectSeq(knownApps,
                    LAMBDA app : app ≠ joinRequest.app))) ;

            UPDATE KNOWN APPS AND PICK NEW GOSSIP FRIENDS
            if joinRequest.app ∉ PT!Range(knownApps)
             then
                knownApps := updatedKnownApps ;
                gossipFriends := PickGossipFriends(self, knownApps) ;
            end if ;

            NOTIFY NETWORK OF NEW JOINER
            notify(gossipFriends −− joinRequest.app, joinRequest.app) ;

            SET STATUS TO JOINED
            joined := TRUE ;
        end with ;
    end if ;
or
    RECEIVE JOIN RESPONSE
    Upon receiving a join response from a gate app, an app will have to update its known
    apps and pick gossip friends.
    await joinRespQueue[self] ≠ ⟨⟩ ;
    await ¬networkOutage ;
    with joinResponse = Head(joinRespQueue[self])
     do
        PULL FROM RESP QUEUE
        joinRespQueue[self] := Tail(joinRespQueue[self]) ;

        UPDATE KNOWN APPS
        knownApps := MergeKnownApps(knownApps, joinResponse.knownHosts) ;

        PICK NEW GOSSIP FRIENDS
        gossipFriends := PickGossipFriends(self, knownApps) ;

        SET STATUS TO JOINED
```

$joined$:= TRUE ;
end with ;
or
RECEIVE JOIN NOTIFICATION
Upon receiving a join notification from any other *app*, an *app* updates its known *apps* as well as its gossips.
await $newJoinerNotif[self] \neq \langle\rangle$;
await $\neg networkOutage$;
with $joinNotifMsg = Head(newJoinerNotif[self])$
do
PULL FROM *NOTIF* QUEUE
$newJoinerNotif[self] := Tail(newJoinerNotif[self])$;
UPDATE KNOWN *APPS*
$knownApps := MergeKnownApps(knownApps, \langle joinNotifMsg.app\rangle)$;
PICK NEW GOSSIP FRIENDS
$gossipFriends := PickGossipFriends(self, knownApps)$;
end with ;

Following are the actions applying to the shopy-list managed by the *app*. We need to abstract actions down to one single add action in order not to have infinite loops between adding, removing and so on. User behavior must not be constrained because it cannot be controlled.
or
ADD
await $Cardinality(shopyList[self]) < Cardinality(PRODUCTS)$;
$shopyList[self] := shopyList[self] ++ NewShopyItem(shopyList[self])$;

Below actions manage the synchronization of the list.
or
SEND *SYNC* REQUEST
with $a \in (PT!Range(knownApps) -- self)$,
$req = NewSyncReq(self)$
do
$takenIDs := takenIDs ++ req.id$;
$syncReqQueue[a] := Append(syncReqQueue[a], req)$;
end with ;
or
RECEIVE *SYNC* REQUEST
await $syncReqQueue[self] \neq \langle\rangle$;
await $\neg networkOutage$;
with $syncRequest = Head(syncReqQueue[self])$,
$mergeResult = shopyList[self] \cup syncRequest.list$,
$newResp = NewSyncResp(self, mergeResult, syncRequest.id)$
do

```
                syncReqQueue[self] := Tail(syncReqQueue[self]) ;
                 merge from request app
                shopyList[self] := mergeResult ;
                syncRespQueue[syncRequest.app] := Append(syncRespQueue[syncRequest.app], newResp) ;
            end with ;
        or
             RECEIVE SYNC RESPONSE
            await syncRespQueue[self] ≠ ⟨⟩ ;
            await ¬networkOutage ;
            with syncResponse = Head(syncRespQueue[self]),
                 mergeResult = shopyList[self] ∪ syncResponse.list
             do

                shopyList[self] := mergeResult ;
                syncRespQueue[self] := Tail(syncRespQueue[self]) ;
            end with ;
        end either ;
    end while ;
end process ;

end algorithm ;
 BEGIN TRANSLATION (chksum(pcal) = "7a780ec0" ∧ chksum(tla) = "e46ebe0c")
VARIABLES isGate, shopyList, syncReqQueue, syncRespQueue, joinReqQueue,
          joinRespQueue, newJoinerNotif, takenIDs

 define statement
NewShopyItem(list) ≜
    [id     ↦ (CHOOSE x ∈ PRODUCTS : ¬∃ i ∈ list : x = i.id),
     bought ↦ FALSE]

ExistingShopyItem(list) ≜ CHOOSE x ∈ list : TRUE

ExistingNotBoughtShopyItem(list) ≜ CHOOSE x ∈ list : x.bought = FALSE



NewSyncMsg(id, a, l, ml, t) ≜
    [id ↦ id,
     app ↦ a,
     list ↦ l,
     mergedList ↦ ml,
     type ↦ t]

NewSyncReqMsg(a, l, ml, t) ≜
    NewSyncMsg(
        (CHOOSE i ∈ IDs : ∀ ti ∈ takenIDs : i ≠ ti),
```

```
        a, l, ml, t
    )

NewSyncReq(app) ≜
    NewSyncReqMsg(app, shopyList[app], {}, REQ_SYNC_ACTION)

NewSyncResp(app, mergeResult, id) ≜
    NewSyncMsg(id, app, shopyList[app], mergeResult, RESP_SYNC_ACTION)



NewJoinReqMsg(app) ≜ [app ↦ app]

NewJoinRespMsg(app, hosts) ≜
    [app ↦ app,
     knownHosts ↦ hosts]

GateApps ≜ {a ∈ APPS : isGate[a]}

NewJoinerNotifReq(app) ≜ [app ↦ app]



PickGossipFriends(app, knownApps) ≜
    LET
        Opposit ≜
            LET I ≜ PT!Index(knownApps, app)IN
            I − (Len(knownApps) ÷ 2) +
            (IF I < Len(knownApps) ÷ 2 THEN Len(knownApps) ELSE 0)

        KnownApps(exclude) ≜
            PT!ReduceSeq(
                LAMBDA a, acc : IF a ∈ exclude THEN acc ELSE Append(acc, a),
                knownApps, ⟨⟩)

        Between(val, min, max) ≜
            PT!Max(min, PT!Min(max, val))

        FirstGossip ≜ LET ka ≜ KnownApps({app})IN
            {ka[Between(Opposit − 1, 1, Len(ka))]}

        SecondGossip ≜ LET ka ≜ KnownApps({app} ∪ FirstGossip)IN
            IF Len(ka) > 0
             THEN {ka[Between(Opposit, 1, Len(ka))]}
             ELSE {}

    IN  FirstGossip ∪ SecondGossip
```

```
MergeKnownApps(apps1, apps2) ≜
  LET AppSeq(n) ≜ PT!SeqOf(APPS, n)

      Contains(appSeq, appItem) ≜
        Cardinality(PT!Matching(appSeq, appItem)) > 0

      f[args ∈ AppSeq(Len(apps1))
               × AppSeq(Len(apps2))
               × AppSeq(Len(apps1) + Len(apps2))] ≜

        LET l1 ≜ args[1]

            l2 ≜ args[2]

            acc ≜ args[3]

            PickFromL1 ≜ f[⟨
                Tail(l1),
                l2,
                Append(acc, Head(l1))⟩]

            SkipOneL1 ≜ f[⟨
                Tail(l1),
                l2,
                acc⟩]

            PickFromL2 ≜ f[⟨
                l1,
                Tail(l2),
                Append(acc, Head(l2))⟩]

            SkipOneL2 ≜ f[⟨
                l1,
                Tail(l2),
                acc⟩]
        IN
          IF Len(l2) = 0
          THEN IF Len(l1) = 0
                 THEN acc
                 ELSE IF Contains(acc, Head(l1)) THEN SkipOneL1 ELSE PickFromL1
          ELSE IF Head(l1) ≠ Head(l2)
                 THEN IF Contains(acc, Head(l2)) THEN SkipOneL2 ELSE PickFromL2
                 ELSE IF Contains(acc, Head(l1)) THEN SkipOneL1 ELSE PickFromL1
  IN  f[⟨apps1, apps2, ⟨⟩⟩]

VARIABLES networkOutage, joined, gossipFriends, knownApps

vars ≜ ⟨isGate, shopyList, syncReqQueue, syncRespQueue, joinReqQueue,
```

$joinRespQueue$, $newJoinerNotif$, $takenIDs$, $networkOutage$, $joined$,
$gossipFriends$, $knownApps\rangle$

$ProcSet \triangleq (APPS)$

$Init \triangleq$ Global variables
$\land\ isGate = [a \in APPS \mapsto a \in GATEAPPS]$
$\land\ shopyList = [a \in APPS \mapsto \{\}]$
$\land\ syncReqQueue = [a \in APPS \mapsto \langle\rangle]$
$\land\ syncRespQueue = [a \in APPS \mapsto \langle\rangle]$
$\land\ joinReqQueue = [a \in APPS \mapsto \langle\rangle]$
$\land\ joinRespQueue = [a \in APPS \mapsto \langle\rangle]$
$\land\ newJoinerNotif = [a \in APPS \mapsto \langle\rangle]$
$\land\ takenIDs = \{\}$
Process *ClientApp*
$\land\ networkOutage = [self \in APPS \mapsto \text{FALSE}]$
$\land\ joined = [self \in APPS \mapsto \text{FALSE}]$
$\land\ gossipFriends = [self \in APPS \mapsto \{\}]$
$\land\ knownApps = [self \in APPS \mapsto \langle self\rangle]$

$ClientApp(self) \triangleq \land \lor \land \neg joined[self]$
$\land \exists\, a \in (GateApps -- self) :$
$joinReqQueue' = [joinReqQueue \text{ EXCEPT } ![a] = Append(joinReqQueue[a], NewJ$
$\land \text{UNCHANGED } \langle shopyList, syncReqQueue, syncRespQueue, joinRespQueue, newJoin$
$\lor \land networkOutage' = [networkOutage \text{ EXCEPT } ![self] = \text{TRUE}]$
$\land \text{UNCHANGED } \langle shopyList, syncReqQueue, syncRespQueue, joinReqQueue, joinResp$
$\lor \land \text{IF } isGate[self]$
$\text{THEN } \land \neg networkOutage[self]$
$\land\ joinReqQueue[self] \neq \langle\rangle$
$\land \text{LET } joinRequest \triangleq Head(joinReqQueue[self]) \text{IN}$
$\text{LET } updatedKnownApps \triangleq Append(knownApps[self], joinRequest.$
$\land\ joinReqQueue' = [joinReqQueue \text{ EXCEPT } ![self] = Tail(joinReq$
$\land\ joinRespQueue' = [joinRespQueue \text{ EXCEPT } ![joinRequest.app] =$

$\land \text{IF } joinRequest.app \notin PT!Range(knownApps[self])$
$\text{THEN } \land\ knownApps' = [knownApps \text{ EXCEPT } ![self] = upda$
$\land\ gossipFriends' = [gossipFriends \text{ EXCEPT } ![self] =$
$\text{ELSE } \land \text{TRUE}$
$\land \text{UNCHANGED } \langle gossipFriends,$
$knownApps\rangle$
$\land\ newJoinerNotif' = [a \in APPS \mapsto$
$\text{IF } a \in (gossipFriends'[self] -- joinRequest.a$
$\text{THEN } Append(newJoinerNotif[a], NewJoine$
$\text{ELSE } newJoinerNotif[a]]$

$\land\ joined' = [joined \text{ EXCEPT } ![self] = \text{TRUE}]$

ELSE $\land$ TRUE

$\land$ UNCHANGED $\langle joinReqQueue, joinRespQueue, newJoinerNotif, joined, gossipFriends, knownApps\rangle$

$\land$ UNCHANGED $\langle shopyList, syncReqQueue, syncRespQueue, takenIDs, networkOutag$

$\lor\ \land\ joinRespQueue[self] \neq \langle\rangle$

$\land\ \neg networkOutage[self]$

$\land$ LET $joinResponse \triangleq Head(joinRespQueue[self])$IN

$\land\ joinRespQueue' = [joinRespQueue \text{ EXCEPT } ![self] = Tail(joinRespQueue[self$

$\land\ knownApps' = [knownApps \text{ EXCEPT } ![self] = MergeKnownApps(knownApps[s$

$\land\ gossipFriends' = [gossipFriends \text{ EXCEPT } ![self] = PickGossipFriends(self, kn$

$\land\ joined' = [joined \text{ EXCEPT } ![self] = \text{TRUE}]$

$\land$ UNCHANGED $\langle shopyList, syncReqQueue, syncRespQueue, joinReqQueue, newJoine$

$\lor\ \land\ newJoinerNotif[self] \neq \langle\rangle$

$\land\ \neg networkOutage[self]$

$\land$ LET $joinNotifMsg \triangleq Head(newJoinerNotif[self])$IN

$\land\ newJoinerNotif' = [newJoinerNotif \text{ EXCEPT } ![self] = Tail(newJoinerNotif[se$

$\land\ knownApps' = [knownApps \text{ EXCEPT } ![self] = MergeKnownApps(knownApps[s$

$\land\ gossipFriends' = [gossipFriends \text{ EXCEPT } ![self] = PickGossipFriends(self, kn$

$\land$ UNCHANGED $\langle shopyList, syncReqQueue, syncRespQueue, joinReqQueue, joinResp$

$\lor\ \land\ Cardinality(shopyList[self]) < Cardinality(PRODUCTS)$

$\land\ shopyList' = [shopyList \text{ EXCEPT } ![self] = shopyList[self] ++ NewShopyItem(shop$

$\land$ UNCHANGED $\langle syncReqQueue, syncRespQueue, joinReqQueue, joinRespQueue, new$

$\lor\ \land\ \exists\, a \in (PT!Range(knownApps[self]) -- self)$:

LET $req \triangleq NewSyncReq(self)$IN

$\land\ takenIDs' = takenIDs ++ req.id$

$\land\ syncReqQueue' = [syncReqQueue \text{ EXCEPT } ![a] = Append(syncReqQueue[a],$

$\land$ UNCHANGED $\langle shopyList, syncRespQueue, joinReqQueue, joinRespQueue, newJoin$

$\lor\ \land\ syncReqQueue[self] \neq \langle\rangle$

$\land\ \neg networkOutage[self]$

$\land$ LET $syncRequest \triangleq Head(syncReqQueue[self])$IN

LET $mergeResult \triangleq shopyList[self] \cup syncRequest.list$IN

LET $newResp \triangleq NewSyncResp(self, mergeResult, syncRequest.id)$IN

$\land\ syncReqQueue' = [syncReqQueue \text{ EXCEPT } ![self] = Tail(syncReqQueue[se$

$\land\ shopyList' = [shopyList \text{ EXCEPT } ![self] = mergeResult]$

$\land\ syncRespQueue' = [syncRespQueue \text{ EXCEPT } ![syncRequest.app] = Appen$

$\land$ UNCHANGED $\langle joinReqQueue, joinRespQueue, newJoinerNotif, takenIDs, network$

$\lor\ \land\ syncRespQueue[self] \neq \langle\rangle$

$\land\ \neg networkOutage[self]$

$\land$ LET $syncResponse \triangleq Head(syncRespQueue[self])$IN

LET $mergeResult \triangleq shopyList[self] \cup syncResponse.list$IN

$\land\ shopyList' = [shopyList \text{ EXCEPT } ![self] = mergeResult]$

$\land\ syncRespQueue' = [syncRespQueue \text{ EXCEPT } ![self] = Tail(syncRespQueue[$

$\land$ UNCHANGED $\langle syncReqQueue, joinReqQueue, joinRespQueue, newJoinerNotif, tak$
$\land$ UNCHANGED $isGate$

$Next \triangleq (\exists\, self \in APPS : ClientApp(self))$

$Spec \triangleq \land Init \land \Box[Next]_{vars}$
$\land \forall\, self \in APPS : \mathrm{SF}_{vars}(ClientApp(self))$

END TRANSLATION

There's no duplicated items in the sequence 'seq'.

$NoDuplicates(seq) \triangleq$
$\forall\, i, j \in$ DOMAIN $seq :$
$i \neq j \Rightarrow seq[i] \neq seq[j]$

All *apps* in 'joinedApps' that joined the network.

$JoinedApps(joinedF, apps) \triangleq \{j \in apps : joinedF[j]\}$

The count of gossips for an *app* 'a' is the number of other joined *apps* that lists 'a' in its gossip friends set.

$CountGossipOf(app, gossips, joinedApps) \triangleq$
$PT!ReduceSet($
LAMBDA $a, acc : acc + ($IF $app \in gossips[a] \land joinedApps[a]$
THEN 1 ELSE $0),$
$APPS, 0)$

The average of gossips is the average count over all joined *apps* of the count of gossips for an *app*.

$AverageGossipOf(gossips, joinedApps) \triangleq$
$PT!ReduceSet($
LAMBDA $a, acc : acc + CountGossipOf(a, gossips, joinedApps),$
$APPS, 0)$
$\div$
$Cardinality(JoinedApps(joinedApps, APPS))$

$ExistsRoute(from, to, \_gossipFriends) \triangleq$
LET $f[\langle app, visited\rangle \in APPS \times$ SUBSET $APPS] \triangleq$
$to \in \_gossipFriends[app]$
$\lor \exists\, a \in (\_gossipFriends[app] \setminus visited) : f[a, visited ++ app]$
IN $from = to \lor f[\langle from, \{\}\rangle]$

$TypeOK \triangleq$
$\land \quad \forall\, a \in APPS :$
Checking on variables' domains.
$\land shopyList[a] \subseteq ShopyItems$
$\land PT!Range(syncReqQueue[a]) \subseteq SyncMsgs$
$\land PT!Range(syncRespQueue[a]) \subseteq SyncMsgs$
The queue for join requests is only for gate *apps*

$\land$ IF $isGate[a]$
THEN $PT!Range(joinReqQueue[a]) \subseteq JoinReqMsgs$
$\land \forall req \in PT!Range(joinReqQueue[a]) : req.app \neq a$
ELSE $joinReqQueue[a] = \langle\rangle$
$\land PT!Range(joinRespQueue[a]) \subseteq JoinRespMsgs$
$\land PT!Range(newJoinerNotif[a]) \subseteq JoinNotifMsgs$
knownApps is a collection of unique, ordered *apps*.
$\land PT!Range(knownApps[a]) \subseteq APPS$
$\land NoDuplicates(knownApps[a])$
Apps don't gossip themselves
$\land gossipFriends[a] \subseteq (APPS -- a)$
$\land \quad takenIDs \subseteq IDs$

$GossipInvariants \triangleq$
$\land \forall a \in APPS :$
Choose more than one gossip if have more than one known *app*.
$\land Cardinality(PT!Range(knownApps[a]) -- a) > 1 \Rightarrow Cardinality(gossipFriends[a]) > 1$
Invariant when we're connected or not.
$\land gossipFriends[a] \neq \{\}$
$\equiv knownApps[a] \neq \langle a \rangle$
We verify that for every joined *app*, the count of any other joined *app* gossiping the new joiner is more or less in the average.
$\land \forall ja \in JoinedApps(joined, APPS) :$
$\lor (\exists n \in JoinedApps(joined, APPS) : newJoinerNotif[n] \neq \langle\rangle)$
$\lor CountGossipOf(ja, gossipFriends, joined) = 0$
$\lor (CountGossipOf(ja, gossipFriends, joined) \geq AverageGossipOf(gossipFriends, joined) - 1$
$\land CountGossipOf(ja, gossipFriends, joined) \leq AverageGossipOf(gossipFriends, joined) + 1$
)

$Liveness \triangleq$
At some point, someone has joined and gossips have been assigned.
$\land \Diamond(\forall ja \in JoinedApps(joined, APPS) : joined[ja] \land CountGossipOf(ja, gossipFriends, joined) > 0)$

$\land \forall a \in APPS :$
Joining leads to having a route from every other joined *app* to the joiner.
$\land joined[a]$
$\leadsto \forall a2 \in JoinedApps(joined, APPS -- a) :$
$ExistsRoute(a2, a, gossipFriends)$
Joining leads to every other joined *app* adding the new joiner to its list of known *apps*.
$\land joined[a]$
$\leadsto \forall a2 \in JoinedApps(joined, APPS -- a) :$
$a \in PT!Range(knownApps[a2])$

* Modification History
* Last modified *Fri Mar* 19 12:44:19 *CET* 2021 by *davd*

\ * Created *Tue Mar* 02 12:33:43 *CET* 2021 by *davd*