MODULE *HKFM*

EXTENDS *Integers*, *Sequences*
CONSTANTS *Client*, *Song*
VARIABLES *inbox*, *state*

Type definitions (kinda sorta) and other useful stuff...

There are various places where we want to refer to all variables at once, so it's useful to define a *vars* tuple.

$$vars \triangleq \langle inbox, state \rangle$$

The constant *Client* is the set of all clients, represented however we want. It's defined externally, in the model.

We define *Node* to be the set of all nodes in the system, including the server. We don't care how the server is represented, only that it doesn't clash with any of the clients. We use the TLA+ CHOOSE operator to express this.

$$\begin{aligned} Server &\triangleq \text{CHOOSE } x : x \notin Client \\ Node &\triangleq Client \cup \{Server\} \end{aligned}$$

These terms all relate to the *playhead*. A *playhead* has two fields:

i: the current track in the *playlist*
t: the number of seconds into that track

When $i = -1$ it means we're not playing anything.

Every node has a *State* consisting of two fields:

playlist: a sequence of songs from the constant set *Song*
playhead: as described above

$$\begin{aligned} Idx &\triangleq Nat \cup \{-1\} \\ Playlist &\triangleq Seq(Song) \\ Playhead &\triangleq [i : Idx,\ t : Nat] \\ Stopped &\triangleq [i \mapsto -1,\ t \mapsto 0] \\ State &\triangleq [playlist : Playlist,\ playhead : Playhead] \\ InitState &\triangleq [playlist \mapsto \langle\rangle,\ playhead \mapsto Stopped] \end{aligned}$$

Clients send "add", "seek", and "skip" messages to the server and the server sends "sync" messages to all clients whenever its state changes. The term *Message* is the set of all possible messages that can occur.

$$\begin{aligned} Message \triangleq\ & [action : \{\text{"sync"}\},\ data : State] \cup \\ & [action : \{\text{"add"}\},\ data : Song,\ sender : Client] \cup \\ & [action : \{\text{"seek"}, \text{"skip"}\},\ data : Playhead,\ sender : Client] \end{aligned}$$

The *TypeOK* formula states that *inbox* must be a function from nodes to sequences of messages and state must be a function from nodes to states. We can ask *TLC* to check that *TypeOK* is an invariant of every behaviour, meaning it will find circumstances where *inbox* and state end up looking wonky. It's also useful to have as a high level type definition for these variables.

```
$TypeOK \triangleq$  $\land\ inbox \in [Node \rightarrow Seq(Message)]$
             $\land\ state\ \ \in [Node \rightarrow State]$
```

Message Constructors

These operators are just for convenience when creating messages in actions below.

```
$SyncMsg \triangleq$
  $[action \mapsto \text{“sync”},\ data \mapsto state'[Server]]$

$AddMsg(client, song) \triangleq$
  $[action \mapsto \text{“add”},\ data \mapsto song,\ sender \mapsto client]$

$SeekMsg(client, playhead) \triangleq$
  $[action \mapsto \text{“seek”},\ data \mapsto playhead,\ sender \mapsto client]$

$SkipMsg(client, playhead) \triangleq$
  $[action \mapsto \text{“skip”},\ data \mapsto playhead,\ sender \mapsto client]$
```

Client Actions

```
$SendAdd(self, song) \triangleq$
  LET
    $msg \triangleq AddMsg(self, song)$
  IN
    $\land\ inbox' = [inbox$ EXCEPT $![Server] = Append(inbox[Server], msg)]$
    $\land$ UNCHANGED $state$

$RecvSync(self) \triangleq$
  $\land\ inbox[self] \neq \langle\rangle$
  $\land$ LET
        $msg \triangleq Head(inbox[self])$
        $tail \triangleq Tail(inbox[self])$
      IN
        $\land\ msg.action = \text{“sync”}$
        $\land\ inbox' = [inbox$ EXCEPT $![self] = tail]$
        $\land\ state'\ = [state$ EXCEPT $![self]\ = msg.data]$

$SendSeek(self) \triangleq$
  LET
    $playhead \triangleq state[self].playhead$
    $msg \triangleq SeekMsg(self, [playhead$ EXCEPT $!.t = playhead.t + 1])$
  IN
     $\land$ $playhead \neq Stopped$
     $\land$ $inbox' = [inbox$ EXCEPT $![Server] = Append(inbox[Server], msg)]$
     $\land$ UNCHANGED $state$

$SendSkip(self) \triangleq$
  LET
    $playhead \triangleq state[self].playhead$
    $msg \triangleq SkipMsg(self, playhead)$
  IN
     $\land$ $playhead \neq Stopped$
     $\land$ $inbox' = [inbox$ EXCEPT $![Server] = Append(inbox[Server], msg)]$
     $\land$ UNCHANGED $state$
```

Server Actions

```
$BroadcastSync \triangleq$
   $\land$ $inbox' = [n \in Node \mapsto$ IF $n = Server$
                                          THEN $Tail(inbox[n])$
                                          ELSE $Append(inbox[n], SyncMsg)]$

$RecvAdd \triangleq$
   $\land$ $inbox[Server] \neq \langle\rangle$
   $\land$ LET
         $server \triangleq state[Server]$
         $msg \triangleq Head(inbox[Server])$
       IN
          $\land$ $msg.action =$ "add"
          $\land$ LET
               $newPlaylist \triangleq Append(server.playlist, msg.data)$
               $newPlayhead \triangleq$ IF $server.playhead = Stopped$
                                    THEN $[i \mapsto Len(server.playlist), t \mapsto 0]$
                                    ELSE $server.playhead$
             IN
               $\land$ $state' = [state$ EXCEPT $![Server] = [playlist \mapsto newPlaylist,$
                                                            $playhead \mapsto newPlayhead]]$
               $\land$ $BroadcastSync$

$RecvSeek \triangleq$
   $\land$ $inbox[Server] \neq \langle\rangle$
   $\land$ LET
         $server \triangleq state[Server]$
         $msg \triangleq Head(inbox[Server])$
       IN
          $\land$ $msg.action =$ "seek"
```

```
        $\land\ state' = [state\ \textsc{except}\ ![Server].playhead.t = msg.data.t]$
        $\land\ BroadcastSync$

$RecvSkip \triangleq$
  $\land\ inbox[Server] \neq \langle\rangle$
  $\land$ LET
      $server \triangleq state[Server]$
      $msg \triangleq Head(inbox[Server])$
    IN
      $\land\ msg.action = \text{“skip”}$
      $\land$ LET
          $newIndex \triangleq server.playhead.i + 1$
          $newPlayhead \triangleq$ IF $newIndex < Len(server.playlist)$
                                  THEN $[i \mapsto newIndex,\ t \mapsto 0]$
                                  ELSE $Stopped$
        IN
          $\land\ state' = [state\ \textsc{except}\ ![Server].playhead = newPlayhead]$
          $\land\ BroadcastSync$
```

Randomly lose a message from an *inbox*

```
$Remove(i,\ seq) \triangleq$
  $[j \in 1 \,..\, (Len(seq) - 1) \mapsto$ IF $j < i$ THEN $seq[j]$ ELSE $seq[j + 1]]$

$LoseMsg \triangleq$
  $\exists\, n \in$ DOMAIN $inbox$ :
    $\exists\, i \in$ DOMAIN $inbox[n]$ :
      $\land\ inbox' = [inbox\ \textsc{except}\ ![n] = Remove(i,\ inbox[n])]$
      $\land$ UNCHANGED $state$
```

Spec

```
$Init \triangleq$
  $\land\ inbox = [n \in Node \mapsto \langle\rangle]$
  $\land\ state\ = [n \in Node \mapsto InitState]$

$Next \triangleq$
  $\lor\ \exists\, self \in Client,\ song \in Song : SendAdd(self,\ song)$
  $\lor\ \exists\, self \in Client : RecvSync(self)$
  $\lor\ \exists\, self \in Client : SendSeek(self)$
  $\lor\ \exists\, self \in Client : SendSkip(self)$
  $\lor\ RecvAdd$
  $\lor\ RecvSeek$
  $\lor\ RecvSkip$
  $\lor\ LoseMsg$
```

$Spec \triangleq$

$\quad Init \land \Box[Next]_{vars}$

THEOREM $Spec \Rightarrow \Box TypeOK$
