LSD Specification for a train arrival-departure protocol

```
/*****************************************************************************
         file     :        train.lsd
         date     :        12/14/94
         author   :        Simon Yun Pui Yung ('modernised' by wmb)
         notes    :        description of agents for a `railway arrival-departure' system
******************************************************************************/
```

The stationmaster

```
agent sm() {

oracle   (time) Limit, Time,                 // knowledge of time to elapse before departure due
         (bool) guard_raised_flag,  // knowledge of whether the guard has raised his flag
         (bool) driver_ready,                // knowledge of whether the driver is ready
         (bool) around[d],          // knowledge of whether there's anybody around doorway
         (bool) door_open[d];       // the open/close status of door d (for d = 1 .. number_of_doors)

state    (time) tarrive = |time|,   // the S-M registers time of arrival
         (bool) can_move,           // the signal observed by driver for starting engine
         (bool) whistle = false,             // the whistle is not blowing
         (bool) whistled = false,   // the whistle has not blown
         (bool) sm_flag = false,             // S-M lowers flag
         (bool) sm_raised_flag = false;      // S-M has not raised flag

handle   (bool) can_move,
         (bool) whistle,
         (bool) whistled,
         (bool) sm_flag,
         (bool) sm_raised_flag;
         (bool) door_open[d];       // the open/close status of door d (for d = 1 .. number_of_doors)

derivate
                           number_of_doors
         (bool) ready =        /\   (!door_open[d]);  // are all doors shut?
                             d = 1
         (bool) timeout = (Time - tarrive) > Limit;   // departure due

protocol
         door_open[d] ^ !around[d] -> door_open[d] = false; (d = 1 .. number_of_doors)
         ready ^ timeout ^ !whistled -> whistle = true; whistled = true; guard(); whistle = false;
         ready ^ whistled ^ !sm_raised_flag -> sm_flag = true; sm_raised_flag = true;
         sm_flag ^ guard_raised_flag -> sm_flag = false;
         ready ^ guard_raised_flag ^ driver_ready ^ engaged ^ !can_move -> can_move = true;

}
```

The guard

```
agent guard() {

oracle   (bool) whistled, sm_raised_flag, brake;

state    (bool) guard_raised_flag = false,
         (bool) guard_flag = false,
         (bool) brake;

handle   (bool) guard_raised_flag, guard_flag;

derivate LIVE = engaging || whistled;

protocol
         engaging -> brake = true; running = false;
         sm_raised_flag ^ brake -> brake = false; guard_flag = true; guard_raised_flag = true;
         guard_flag ^ !sm_flag -> guard_flag = false;

}
```

The driver

```
agent driver() {

oracle   (bool) can_move, engaged, whistled;
         (position) at, from;

handle   (position) to, from,
         (bool) running,
         (bool) driver_ready = false;

state    (bool) driver_ready = false,
         (position) from;

protocol
         whistled ^ !driver_ready -> driver_ready = true;
         engaged ^ from <> at -> from = at; to = next_station_after(at);
         can_move ^ engaged -> driver_ready = false; running = true;

}
```

The passenger

```
agent passenger((int) p, (int) d, (position) _from, (position) _to) {
// passenger p intending to travel from station _from to station _to
// and he will access through door d of the train

oracle   (position) at,
         (bool) door_open[d];

state    (bool) pos[p] = OUT_DOOR, alighting[p], boarding[p], join_queue[p,d];

handle   (position) from[p] = _from;
         (position) to[p] = _to;
         (int) door[p] = d;
         (bool) pos[p],
         (bool) door_open[d];

derivate
         alighting[p] = at == to[p] ^ pos[p] != OUT_DOOR && engaged;
         boarding[p] = at == from[p] ^ pos[p] != IN_DOOR && engaged;
         join_queue[p,d] = (alighting[p] ^ door_open[d] ^ pos[p] == IN_DOOR) ||
                  (boarding[p] ^ door_open[d] ^ pos[p] == OUT_DOOR);
         LIVE = !(at == to[p] ^ pos[p] == ON_PLATFORM);

protocol
         at == to[p] ^ pos[p] == AT_SEAT -> pos[p] = IN_DOOR;
         alighting[p] ^ !door_open[d] -> door_open[d] = true;
         alighting[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ !queuing[d]
                  -> pos[p] == OUT_DOOR; door_open[d] = false; pos[p] = ON_PLATFORM;
         alighting[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ queuing[d]
                  -> pos[p] == OUT_DOOR; pos[p] = ON_PLATFORM;
         boarding[p] ^ !door_open[d] -> door_open[d] = true;
         boarding[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ !queuing[d]
                  -> pos[p] = IN_DOOR; door_open[d] = false; pos[p] = AT_SEAT;
         boarding[p] ^ pos[p] == AT_DOOR ^ door_open[d] ^ queuing[d]
                  -> pos[p] = IN_DOOR; pos[p] = AT_SEAT;

}
```

The train

```
agent train() {

state    (bool) running = true,
         (bool) brake = false,
         (bool) door_open[d] = false, (d = 1 .. number_of_doors)
         (position) from = station1,
         (position) to = station2,
         (position) at = some_position,
         (bool) engaging, engaged, leaving, alert;

handle   (bool) alert;

derivate
         (bool) engaging = running ^ to == at,
         (bool) leaving = running ^ from == at,
         (bool) engaged = !running;

protocol
         engaging ^ !alert -> alert = true; guard(); sm();
         leaving ^ alert -> alert = false; delete guard(), sm();

}
```

The door

```
agent door((int) d) {

oracle   (int) pos[p], door[p]; (p = 1 .. number_of_passengers)

state    (bool) queuing[d], occupied[d], around[d];

derivate
         queuing[d] = there exists p such that join_queue[p,d] == true;
         occupied[d] = there exists p such that (pos[p] == AT_DOOR ^ door[p] == d)
         around[d] = there exists p such that (door[p] == d ^
                  (pos[p] == IN_DOOR || pos[p] == AT_DOOR || pos[p] == OUT_DOOR))

protocol queuing[d] ^ !occupied[d] ^ join_queue[p,d] == true
                  -> pos[p] = AT_DOOR; (p = 1 .. number_of_passengers)
}
```