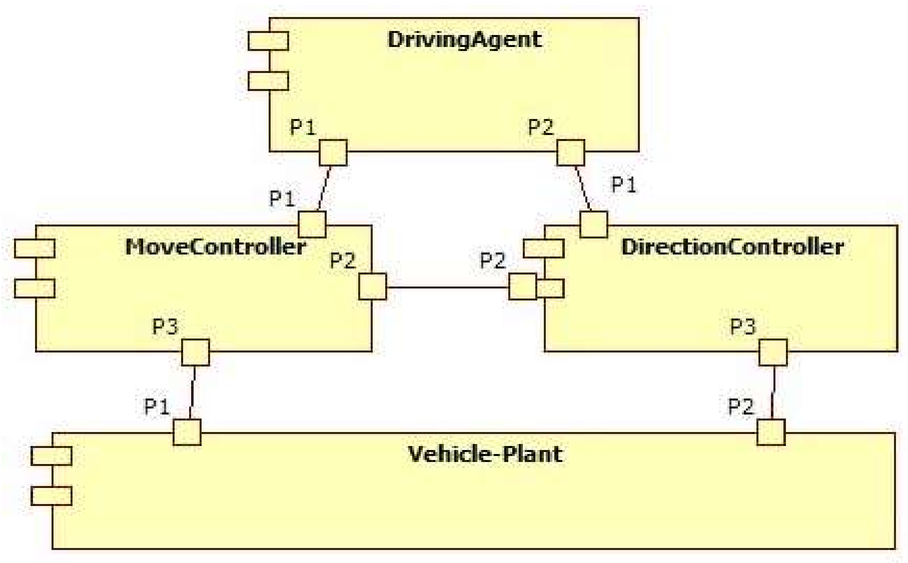
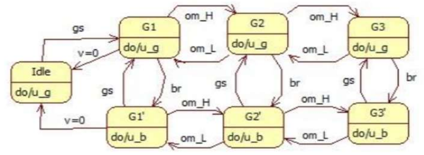
10.10.5 Controlul traficului pentru vehicule autonome – Traffic control of autonomous vehicles

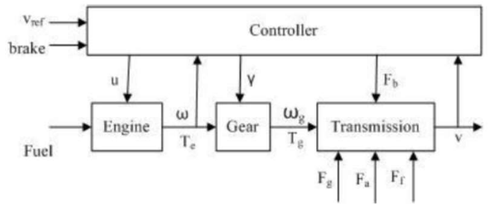
Self-driving vehicle

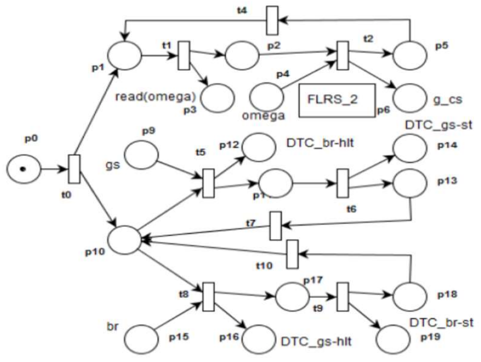
See Ch. 4.5



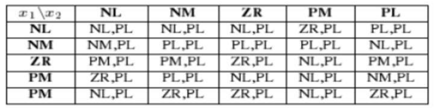
Example: Independent vehicle move control



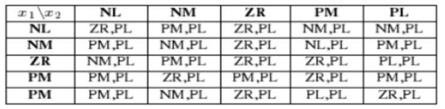




Inference rules for the gas and break cases



$x_1 \backslash x_2$	NL	NM	ZR	PM	PL
NL	NL,PL	NL,PL	NL,PL	ZR,PL	PL,PL
NM	NM,PL	PL,PL	PL,PL	PL,PL	NL,PL
ZR	PM,PL	PM,PL	ZR,PL	NL,PL	PM,PL
PM	ZR,PL	PL,PL	NL,PL	NL,PL	NM,PL
PM	NL,PL	ZR,PL	ZR,PL	NL,PL	ZR,PL



$x_1 \backslash x_2$	NL	NM	ZR	PM	PL
NL	ZR,PL	PM,PL	ZR,PL	NM,PL	NM,PL
NM	PM,PL	NM,PL	ZR,PL	NL,PL	PM,PL
ZR	NM,PL	PM,PL	ZR,PL	ZR,PL	PL,PL
PM	PM,PL	ZR,PL	PM,PL	ZR,PL	PM,PL
PM	PM,PL	NM,PL	ZR,PL	PL,PL	ZR,PL

59

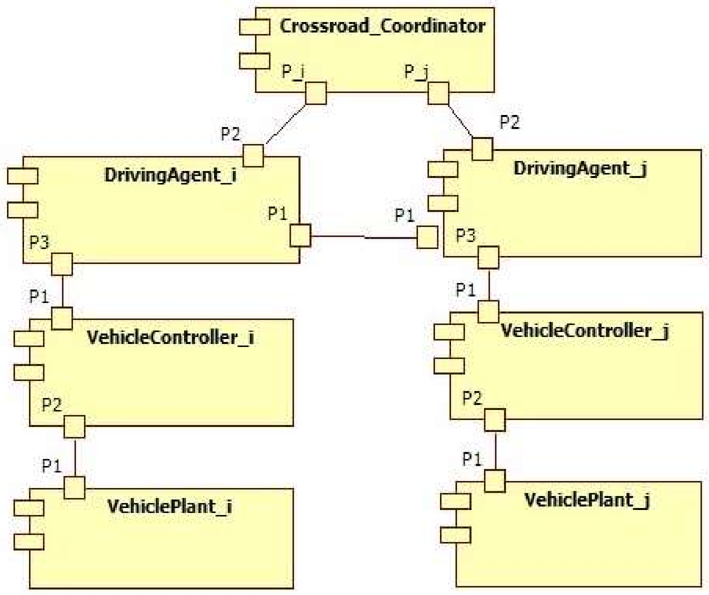
What should be the specifications of:

- Driving agent?
- MoveController?
- DirectionController?

Why have to cooperate MoveController with DirectionController?

The existence of traffic lights are not compulsory.

With coordinator



Each vehicle has a *driving agent* that can communicate with an intersection coordinator (server).

All the vehicle approaching an intersection demand (sending messages to coordinator) the permission to cross specifying the exit lines. The coordinator keeps the requests and grants the crossing taking into account a fair schedule.

The coordinator continuously monitors the vehicles using their GPS and local (crossroad) controller sensors/detectors for the current position.

Without coordinator

The vehicle agents negotiate the crossing using a fair algorithm.

What crossing algorithm would be fair in your opinion?

10.10.6 Monitoring of Independent Mobile (moving) Entities

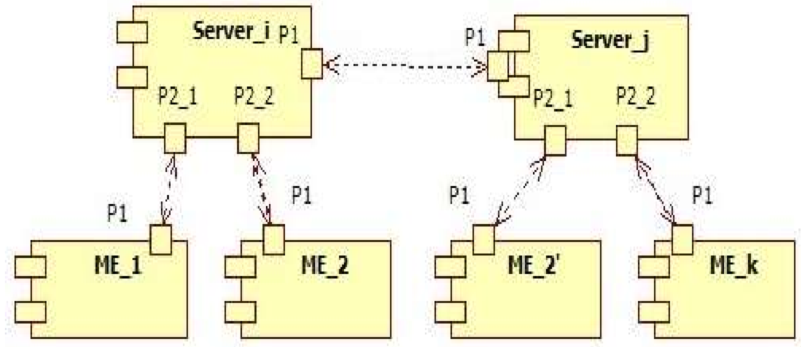
A mobile entity system can include:

- Cars (vehicles)
- Busses
- Trains
- Underground trains
- Robots in a warehouse
- Drones etc.

Moves:

- 2 D (dimension)
- 3 D.

Fig. 1 Moving entities component diagram



. Fig. 1 shows the component diagram of the example used for analysis. This consists of some Moving Entities (MEs) and a set of interconnected servers used for their monitoring. The system development through different stages is described using OETPN models.

A set of ME $\mu_{i,k}$ (i=1,2, …; k=1,2, …) assigned (for the first time i.e. home server) to a zone Z_i (i=1,2, …) are monitored by a set of servers S_i (i=1,2, …).

The entities $\mu_{i,k}$ can move freely in any zone Z_j (j=1,2,…). When $\mu_{i,k}$ is detected in a zone Z_j ($j \neq i$), the server S_j is signaled, and it signals further the server S_i where $\mu_{i,k}$ is assigned.

The entities $\mu_{i,k}$ can asynchronously produce events (denoted by $e_{i,k,x}$) that modify their current state ($\sigma_{i,k}$) and that are signaled to the server of the zone where they are currently positioned. A set C_k (k=1,2, …) of clients can asynchronously interrogate the server S_i where $\mu_{i,k}$ is assigned about the entity current state. Fig. 4 shows the OETPN model of the communicating server.

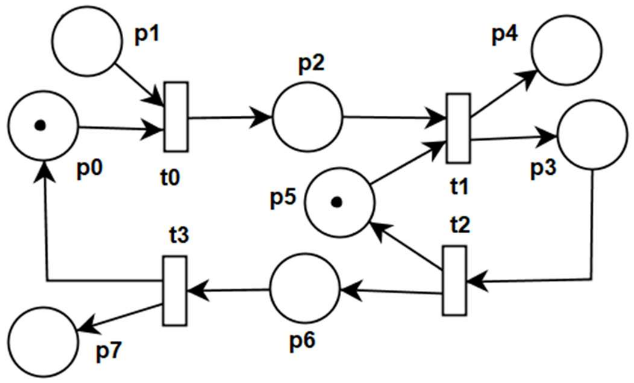
Mobile Entity

When a ME modeled by the OETPN presented in fig. 2 is moving or is moved, it generates automatically events and tries to connect the zone's server sending specified information.

A ME has one input channel (i.e. p1) and two output channels (i.e. p4 and p7).

The place p5 serves for storing the entity identifier, the zone where it is registered and state information like *current position*, etc. When the entity enters in a new zone, it receives through p1 the zone identifier and sends to the local server through p4 its public state information. Other information can be sent to its own home server by the server where ME is currently a guest.

Fig. 2 OETPN Model of Moving Entity



The place types of ME are: *type*(*p1*) = *type*(*p2*) = *type*(*p4*) = *type*(*p6*) = *type*(*p7*) = String; *type*(*p3*) = *type*(*p5*) = List.

The following notations adopted from software are used:

- M(p_i) represents the reference to the object assigned to the place p_i.
- x_i is the principal variable stored in the object referred by M(p_i). When more than one variable is involved, they are denoted by x^1_i, x^2_i, …. Similar notations are used for the guards (grd_i) and mappings (map_i) assigned to transitions.
- M(p_i) = ϕ means M(p_i) = *null* and this can be denoted by $x_i = \phi$. All of these notations describe that the corresponding object is missing at the current moment, so the value of the variable x_i is not available. The opposed notation is M(p_i) $\neq \phi$ or $x_i \neq \phi$.

- M(p_i) = M(p_j) means the object referred by the place p_i is transferred to the place p_j. The equivalent notation is $x_i = x_j$.

The ME's guards and mappings are:

- grd_0: $(M(p0) \neq \phi)$ *AND* $(M(p1)) \neq \phi)$;

  map_0: $M(p2) = M(p1)$; *i.e.* $x_2 = x_1$.
- grd_1: $(M(p2) \neq \phi)$ *AND* $(M(p5)) \neq \phi)$;

  map_1: $M(p3) = M(p3)$ .add($M(p2)$);

  $M(p3)$.add($M(p5)$.*privateInfo*); *i.e.* $x_3 = x_2$,

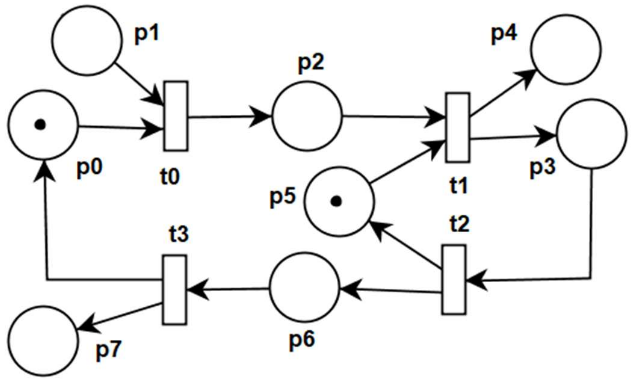
  $M(p4) = M(p5)$.*publicStateInfo*.
- grd_2: $(M(p3) \neq \phi)$;

  map_2: $M(p5) = M(p5)$.currentPos = $M(p3)$.get($M(p2)$).; *i.e.* $x_3 = x_2$,

  $M(p6) = M(p3)$.*privateInfo*.
- grd_3: $(M(p6) \neq \phi)$;

  map_3: $M(p7) = M(p6)$; *i.e.* $x_7 = x_6$,

  $M(p0) = M(p6)$; *i.e.* $x_0 = x_6$.



Server

A server receives events from the MEs positioned in its assigned zone.

The server requests the ME to identify itself and to provide its current state. The server stores this information in its local databases. The server discerns if the ME is a local one or a foreigner ME. For a foreigner ME the server sends to ME's first registration server the ME's current state. A server can receive from another server information related to the state of a ME assigned to the zone which it monitors.

A local client can ask the server about a ME's home server. The local client model is similar to the previous one.

Fig. 3. The Server OETPN Model

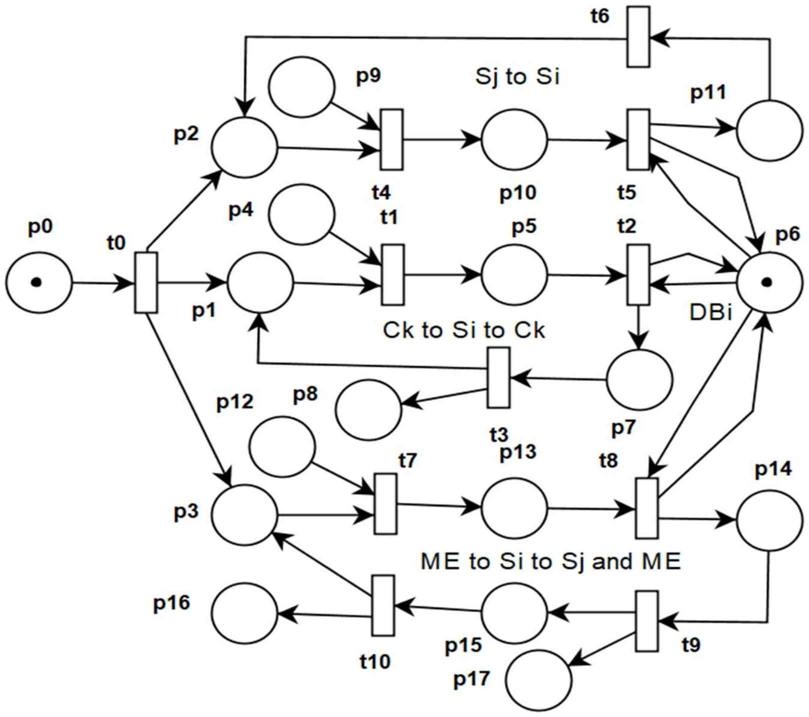


Fig. 3 shows the OETPN model of the server Si. It contains three tasks marked with *Sj to Si*, *Ck to Si to Ck* and *ME to Si* respectively. All of them access the databases DBi referred through the place p6.

The task *Sj to Si* receives information from a server Sj through input channel p9 and store it to DBi. This information concerns the states of MEs first registered in Zi.

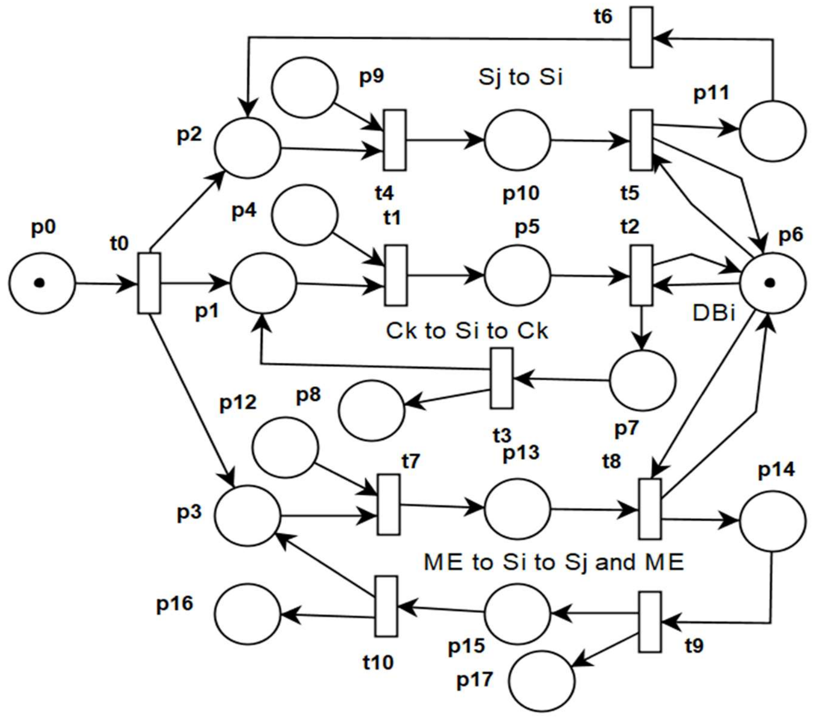
The tasks *Ck to Si to Ck* receives demand (information about a ME's home server) from a client Ck through the input channel p4, it accesses DBi and sends answers related to the history state of ME through its output channel p8.

The task *ME to Si and ME* receives information related to the identifier and the current state of a ME through the channel p12. Si stores this information in the databases DBi. Further, Si discerns if the ME is a local first registered, or it is a foreigner. In the last case, Si sends to the server Sj (ME's home server) information related to ME current state through the output channel p17. Si informs the ME related to the zone where it is currently positioned through the cannel p16.

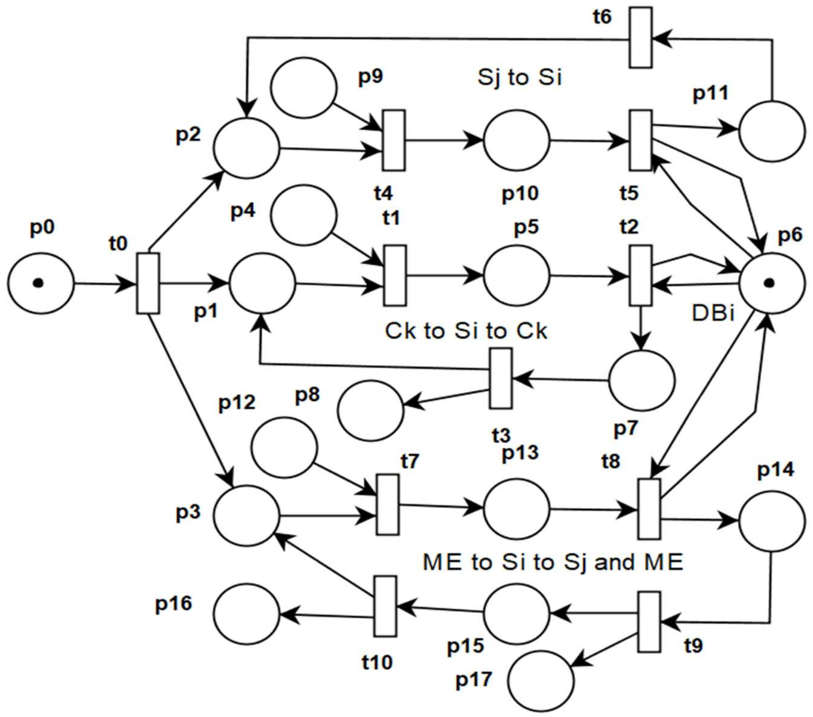
The place types of the server are: $type(p0) = type(p1) = type(p2) = type(p3) = type(p4) = type(p5) = type(p7) = type(p9) = type(p10) = type(p11) = type(p12) = type(p13) = type(p14) = type(p15) = type(p16) = type(p17) =$ String; $type(p6) =$ Database; $type(p8) =$ list;

The server's guards and mappings are:

- grd_0: $(M(p0) \neq \phi)$;
  - map_0: $M(p1) = M(p0)$; $i.e. x_1=x_0$.
  - map_0: $M(p2) = M(p0)$; $i.e. x_2=x_0$.
  - map_0: $M(p3) = M(p0)$; $i.e. x_3=x_0$.
- grd_1: $(M(p1) \neq \phi)$ *AND* $(M(p4)) \neq \phi)$;
  - map_1: $M(p5) = M(p4)$; $i.e. x_5=x_4$.



- grd_2: $(M(p5) \neq \phi)$ *AND* $(M(p6)) \neq \phi)$;
  map_2: $M(p6)$ = queryRequest $(M(p5))$.
  map_2: $M(p7)$ = queryResponse ().
- grd_3: $(M(p7) \neq \phi)$;
  map_3: $M(p8) = M(p7)$; $i.e. x_8=x_7$.
  map_3: $M(p1) = M(p7)$; $i.e. x_1=x_7$.
- grd_4: $(M(p2) \neq \phi)$ *AND* $(M(p9)) \neq \phi)$;
  map_4: $M(p10) = M(p9)$; $i.e. x_{10}=x_9$.
- grd_5: $(M(p10) \neq \phi)$ *AND* $(M(p6)) \neq \phi)$;
  map_5: $M(p6)$ = update.MEpublicInfo $(M(p10))$.
  map_5: $M(p11) = M(p10))$; $i.e. x_{11}=x_{10}$.
- grd_6: $(M(p11) \neq \phi)$;
  map_6: $M(p1) = M(p11)$; $i.e. x_1=x_{11}$.
- grd_7: $(M(p3) \neq \phi)$ *AND* $(M(p12)) \neq \phi)$;
  map_7: $M(p13) = M(p12)$; $i.e. x_{13}=x_{12}$.
- grd_8: $(M(p13) \neq \phi)$;
  map_8: $M(p6)$ = update.MEprivateInfo $(M(p5))$.
  map_8: $M(p6)$ = if (query.MEisforigner()) then
  $M(p14)$ = List.add (query.MElocalPosition.
  $M(p14)$ = List.add (query.MEcurrentPosition().
- grd_9: $(M(p14) \neq \phi)$;



map_9: $M(p15) = M(p14).get(MEcurrentPosition)$.

Map_9: $M(p17) = M(p14)$.get(MElocalPosition).

- grd_{10}: $(M(p15) \neq \phi)$;

  map_{10}: $M(p16) = M(p15)$;

  map_{10}: $M(p3) = M(p15)$;

10.10.7 Self Driving of Mobile Entities

2D Move (X-Y)

The problem is simplified considering a set of autonomous mobile robots (i.e. MEs) that move in a warehouse (2 D) with the *chess board architecture* (see the attached table).

Between blocks are lanes with one robot capacity.

An intersection (cross of the paths) is referred by the block upper left corner.

A robot can move: *left*, *right, forward* or *backward*. The robot move on a block edge (lane) needs 10 sec. duration.

y8								
y7								
y6								
y5								
y4								
y3								
y2								
y1								
	x1	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**

A robot receives (from warehouse *Operator)* tasks to move from one intersection to another (e.g. *(x(k), y(k))*$\rightarrow$*(x(k+1), y(k+1))*. Only one step at each tick. The move on X or on Y.

A robot software is composed of Robot Agent (RA) and Robot Move Controller (RMC).

A RA can communicate with all the RAs, with warehouse Operator and with Coordinator (if there is one).

RA conceives a path corresponding to the received task and sends it to Controller that implements it. A path is a sequence e.g. $\sigma = (3,2)*(3,3)*(3,4)*(3,5)*(4,5)*(4,6)*(4,7)$.

A set of the current paths are feasible if they do not have overlapping segments in the same time.

$\boldsymbol{\theta} \in [0, 2\pi];\ \boldsymbol{\theta} = \boldsymbol{k} \cdot \frac{\pi}{4}; \boldsymbol{k} \in \{0, 1, 2, 3\}$

The classic computation deterministic walker has the model:

$$\begin{bmatrix} o_x(\tau+1) \\ o_y(\tau+1) \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} o_x(\tau) \\ o_y(\tau) \end{bmatrix}$$

$$\begin{bmatrix} x(\tau+1) \\ y(\tau+1) \end{bmatrix} = \begin{bmatrix} x(\tau) \\ y(\tau) \end{bmatrix} + \begin{bmatrix} o_x(\tau+1) \\ o_y(\tau+1) \end{bmatrix}$$

ME moves:

- 10 seconds for one step on each edge. ME can move only on edge.
- Only one ME can be on each lane (including intersection).
- Each ME gets a mission: move from departure=(x_d, y_d) to arrival=(x_a, y_a).

Homework for the above problem

1. Conceive an algorithm (for RA) that constructs paths (i.e. mission) from a start (departure) intersection to a destination (arrival) intersection.
2. Conceive a static resource allocation method (for a fixed set of RAs) that reserves the resources for a given set of paths when there is a centralized allocation Coordinator. Describe it using a set of OETPN models for RAs and an OETPN model for Coordinator. Endow the models with the needed input/output channels.
3. Conceive a dynamic resources allocation method that reserves the resources for a given set of RAs that dynamically demand path reservations when there is a centralized allocation Coordinator.
4. Conceive a static resource allocation method (without coordinator) where the paths (resources) reservations are performed by RAs agreement. Describe it using a set of OETPN models for RAs (see Ch. 6.4 and 6.5).
5. Conceive a dynamic resource allocation method (without coordinator) where the paths (resources) reservation is performed by RAs agreement. Describe it using a set of OETPN models for RAs (see Ch. 6.4 and 6.5).

Add the OETPN model of RMC.
How can be implemented such applications?

ME 3D Move (X-Y-Z) = Flying Drones

The position is given by $(x(k), y(k), z(k)) \rightarrow (x(k+1), y(k+1), z(k+1))$.

ME moves:

- 10 seconds for one step on each edge. ME can move only on edge.
- Only one ME can be on each lane (including intersection).
- The Hight nominal level is ***z=4***. ME usually moves at the ***nominal level***.
- Ascending or descending can be performed in a plane X-Z or Y-Z at an angle of 45^o.
- Crossing the path (i.e. ***conflict***): each ME has an index (natural number). When two ME has to move on an edge (or intersection), they have to change their height according to the rule: ME with the lower index, changes $z(k+1)=z(k)-1$; ME with the greatest changes $z(k+1)=z(k)+1$.

Each ME gets a mission: move from *departure*$=(x_d, y_d, 0)$ to *arrival*$=(x_a, y_a, 0)$ (from ground to ground).

Notice: the orientation formula remain the same as for 2D if X-D is changed with X-Z or Y-Z.

Homework for the above (3D) applications

1. Each ME receives a mission achieved by the ME moveController. Conceive an algorithm for autonomous (self-control) move control that solves the path (channel) conflicts. Conceive the application for two MEs that move autonomously avoiding the collision.

2. Conceive a Coordinator that guides the MEs sending un-conflict paths without intersection conflicts.

*

****

END

****

*