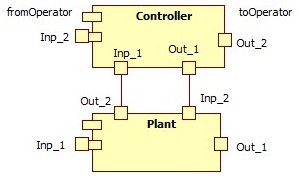
**Synthesis of a controller for reactive applications**



**The synthesis problem**

The specifications are based on the OETPNP model of the plant, term used here as a generic name. The *plant* can be: a vehicle, a tool, a device, or even another software application. The OETPNP model describes the Plant's behavior in a specified initial state and interactions with the outside of the plant.

It is specified for the Plant:

* Inp\_1 input channels that create uncontrolled and unknown events or data streams (by the Controller) in general; the types of this information must be compatible with the places of the plant; sometimes human operators can interact with the installation through some of the input channels in the Inp\_1 set;
* Out\_1 output channels that generate events and data streams outside the plant, but which are not (generally) measured directly (or at least not all) by the Controller; sometimes human operators can receive information about the plant through channels in the Out\_1 set;
* Out\_2 output channels provide events and data streams that can be retrieved by the Controller; their types must be compatible with the corresponding types of the places in the Controller;
* Inp\_2 input channels provide (controllable) information to the plant; they must be specified if they are events or data streams and their types are specified;

The *controller* here means a program that interacts with the Plant and determines the latter to have an evolution required by the specification requirements.

*Specification requirements* describe how the Plant should behave. It is possible for a human operator to interact with the Controller or even the Plant during evolution.

From the specification requirements, the OETPNR model (or a set of models) is constructed, which formally describes the required behavior of Plant.

Examples of requirements are:

* If the Plant is in a specified state and if the inputs receive specified information (events or data streams), then the Plant must evolve on a certain trajectory required by the states, or avoid a state, possibly a sequence of states;
* If the Plant is in a certain state and a given event occurs, the Plant must act according to certain requirements, such as producing an event or sequence of events;

*Controller specifications include*:

* Inp\_1 input channels through which information can be received from the Plant; they must be compatible with the output channels of the Plant; they can be of event type or data streams;
* Out\_1 output channels through which the Controller can influence the behavior of Plant; they provide event type information or data streams; they must be compatible with the Inp\_2 channels in the Plant;
* Inp\_2 channels are used for taking orders from the operator in the form of events or data streams;
* Out\_2 channels provide information to the operator in the form of events or data streams; capable devices are required to retrieve or display the type of the information provided.

The formal problem of the synthesis of the controller is to determine the model of the OETPNC controller and its initial state for the case where OETPNP is in the MP0state, and Plant's behavior corresponds to the OETPNR requirements with MR0.

For some problems, the external inputs for the Plant and the Controller are specified. The former are sometimes referred to as disturbances, and the latter often express the operator's instructions or commands.

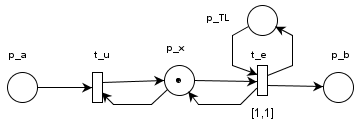
**Example: Synthesis of vehicle traffic control through an intersection**



The infrastructure consists of 4 input lanes (ILi; i=1,2,3,4), 4 output lanes (OLi; i=1,2,3,4), 4 traffic lights (TLi, i=1,2,3,4) and 4 pairs of sensors αi, βi that can measure the inputs and outputs of the corresponding input lanes respectively. Sensors γi (i=1,2,3,4) can measure the inputs to the output lanes.

TLi has 3 states: red (r), yellow (y) and green (g). Cars can only pass when the traffic light indicates the color g. The color duration y is 5 seconds (sec.). The rule of intersection is 'only forward or right'.

**Event-based Plant model**

*The model of a lane* is: xc = xa + u – e

Where xc and xa represents the number of cars on the current and previous lane respectively, and u and e are the events when the entry / exit of a car is signaled. u and e have the value one, only when the respective sensor signals the passage of the machine, and the rest have zero value. The event model of the lane is represented in the attached figure.

Notations:

* p\_a the input channel from the input sensor αi,
* p\_b the output channel from the output sensor βi,
* t\_u the transition that takes over the input event and add car objects to the array list in p\_x
* p\_TL the input channel through which TL commands are received (red, yellow, green)
* t\_e the transition that models (and determines) the output of a car from the lane

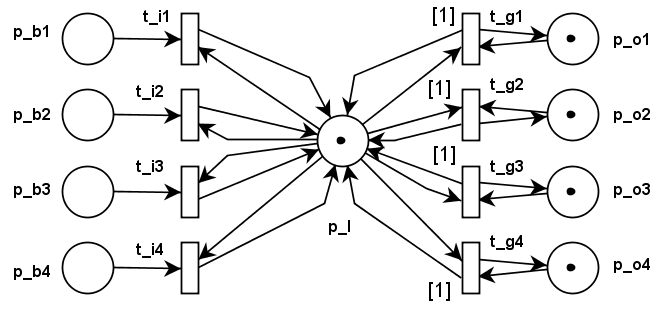
The Place types are:

* type(p\_a) = car object containing list of string to represent target transitions for the car route
* type(p\_x) = arraylist of car with a dynamic size that is set by the user
* type(p\_TL) = string = {red, yellow, green}
* type(p\_b) = car object

The evolution rules associated with the transitions are:

* t\_u: grd: (m(p\_a) ≠ ϕ And m(p\_x).CanAddCars)); map\_u: list.add m(p\_a);
* t\_e: grd: (m(p\_TL) = green And m(p\_x).HaveCar()); map\_e: m(p\_b) = PopCar() (m(p\_x));

where the method CanAddCars() checks if there is a available space in the arraylist, and the method HaveCar() pops a car form the arraylist to the desired place.

*****Intersection model:*

The paths of the open lanes in a phase cannot intersect. The system has two phases marked with: phase\_1 (for IL\_1 and IL\_3) and phase\_2 (for IL\_2 and IL\_4).

The attached model describes the behavior of the intersection.

The nodes meanings are:

* the exit places from the lanes at the intersection
* p\_b1, p\_b2, p\_b3, p\_b4
* the entrance transitions at the intersection t\_i1, t\_i2, t\_i3, t\_i4
* p\_I models cars at the intersection at the current moment
* transitions t\_g1, t\_g2, t\_g3, t\_g4 delayed by 1 sec. Model the input events in the output lanes of the intersection
* the places p\_o1, p\_o2, p\_o3, p\_o4 model the cars leaving the exiting lanes

The place types are:

* type(p\_b1) = type(p\_b2) = type(p\_b3)=type(p\_b4)= car object;
* type(p\_o1) = type(p\_o2) = type(p\_o3) = type(p\_o4) = type(p\_I) = array list of car with a dynamic size that is set by the user

The evolution rules associated with the transitions are:

* t\_i1, ..., t\_i4: (grd\_b1: ((m(p\_b1) ≠ ϕ) And (m(p\_I).CanAddCars ())); map\_i: list.add m(p\_b));
* t\_g1, ..., t\_g4: (grd\_g1: (( (m(p\_I).HaveCarForMe())And (m(p\_o).CanAddCars ())); map\_o1: PopCar() (m(p\_I), AddCar() (m(p\_g1));

where the method HaveCarForMe() checks if the transition t\_g1 is one of the target transitions in the car object.

**The plant model based on data streams**

Unlike the previous model that takes into account the events produced by cars, this model uses data streams. Input channels provide data streams.

In this case, the actual process of the car traffic system is regarded as vehicle flows that are modeled as data streams. In the actual process the vehicle flow is controlled, and in the model the data stream is controlled. The correspondence between the two types of flows is made by the input / output devices. There are sensors that can be read periodically or that can automatically provide vehicle flow parameters. These are generators of information for the controller who will calculate on their basis the commands that are transmitted to the traffic lights and thus control the vehicles flow.

Data streams has a frequency and a data type. Reading and writing in the stream is done with a specified period δ and there is a certain type (structure) of data (eg: real, integer) that are accepted.

The model of the previous lane can be transformed into a model based on flows. It is assumed that the sensor supplies with the period of 1 sec, the number of vehicles entered on the lane, and the sensor β on the one of the vehicles that came out in the same period of 1 sec. The palce p\_x will store the number of vehicles on the lane at that time.

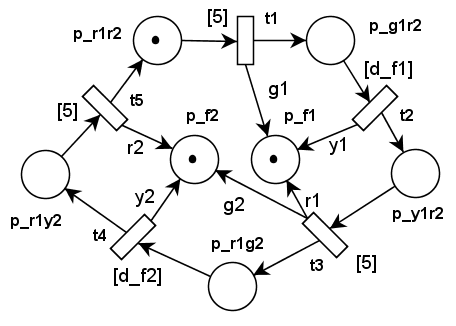
The mathematical model made is:

x(τ+1) = x(τ) + u(τ) – e(τ)

with τ noted as time.

There is the condition *max ≥ x(τ)*.

While u(τ) is an uncontrollable values, output e(τ) can be controlled by the value entered in the place p\_TL. There is the real condition that the number of vehicles at a given time does not exceed the maximum number of vehicles that represents the lane capacity.

**The model of the fixed phase controller of the phases**

For the controller synthesis it is required that the Plant allows the intersection periodically with the specified period divided between the two phases with the durations d\_f1 and d\_f2, for the change of phases the duration of the yellow color to be 5 sec. And the duration of red everywhere to be 5 sec.

The attached figure models the fixed-duration controller of the phases marked on the figure with d\_f1 and d\_f2.

The nodes meanings are:

* p\_r1r2 is the place that expresses the red traffic lights everywhere (clearance red)
* p\_g1r2 specifies the green color for phase 1 and red for phase 2
* p\_y1r2 specifies the yellow color for phase 1 and red for phase 2
* p\_r1g2 has a token when phase 1 is red and phase 2 is green
* p\_r1y2 has a token if phase 1 is red and phase 2 is yellow
* Transitions t1, t2, ..., t5 marked on the colors the colors they impose on the two phases

Transitions t1, t3, t5 have delays of 5 sec each.

The place p\_f1 and p\_f2 are output channels that must be connected to traffic lights p\_TL1, ..., p\_TL4 at the intersection. Initially these channels are red, and the controller is in its original state p\_r1r2.

Type(p\_r1r2) = Type(p\_g1r2) = Type(p\_y1r2) = Type(p\_r1g2) = Type(p\_r1y2) = String;

Type (p\_f2) = type (P\_f2) = DataOverNetwork; that contains the token and a string for the IP address and an integer for the port number.

The guard for transition:

* t1: (grd\_1: ((m(r1r2) ≠ ϕ); map\_f1: SendOverNetwork(green); map\_g1r2 = map\_r1r2);
* t2: (grd\_2: ((m(g1r2) ≠ ϕ); map\_f1: SendOverNetwork(yellow); map\_y1r2 = map\_ g1r2);
* t3: (grd\_3: ((m(y1r2) ≠ ϕ); map\_f1: SendOverNetwork(red); map\_f2: SendOverNetwork(green) ); map\_r1g2 = map\_ y1r2);
* t4: (grd\_4: ((m(r1g2) ≠ ϕ); map\_f2: SendOverNetwork(yellow) ); map\_r1y2 = map\_ r1g2);
* t5: (grd\_1: ((m(r1y2) ≠ ϕ); map\_f2: SendOverNetwork(red) ); map\_r1y2 = map\_ r1r2);

**Controller model with variable phase durations (closed-loop controller)**

The open-loop controller does not consider changing vehicles flows for which the phase durations were calculated.

The variable duration controller must adapt them according to the information received from the sensors (or detectors). Sensors can provide vehicles entry or exit events, vehicles entry and exit rates, so vehicle flows or can measure (estimate) the lengths of queues for waiting vehicles.