Two minutes example

Firstly, we are going to write a simple COOPN class to represent a machine with state "stopped" or "running", its equivalent of classical Petri net likes the following graph:

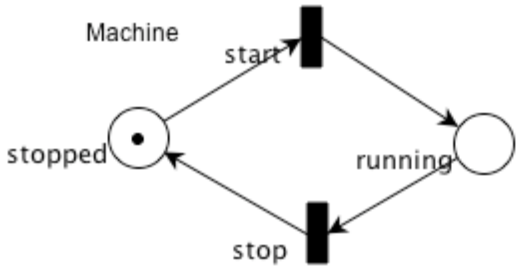

Figure 1.

This trivial Petri net contains two places and two transitions, where the transitions move the state of the machine from one to another. It's a "State machine" because only and always one token exists in the net.

Supposing you've already create a package in CoopnBuilder, just right-click the "Coopn Source" on the package tree, a popup menu with selection "Add module" will appear, by selecting it you can add a module "Machine" in your package. After that you can use the text editor or the tree-like editor to write your COOPN codes.

Follwing are complete source of COOPN class "machine", bold words are keywords of COOPN language:

```
Class machine;

Interface
    Type
        machineType;
    Methods
        start;
        stop;
Body
    Use
        BlackTokens;
    Places
        stopped _ : blackToken;
        running _ : blackToken;
    Initial
        stopped @;
    Axioms
        start::stopped @ -> running @;
        stop::running @ -> stopped @;
End machine;
```

In this class, we have two visible methods: start and stop, two places which represent the state of the machine: stopped_ and running_, the valid value type of the places is blackToken, this means the places can only have tokens of type "blackToken".

The initial marking is a token in the place "stopped_", we've also defined two axioms to specify the pre- and post-condition of the two transition. The axiom

```
start::stopped @ -> running @
```

says that the pre-condition of transition start is a token "@" in the place "stopped_", after the transition, the token is moved to the place "running_".

You'll see that we "Use" the module "BlackTokens", the keyword **Use** has the same meaning of "include" or "import" of other programming languages. The module "BlackTokens" is an ADT, it defines only one symbol "@" representing the non-typed token of Place/Transition nets.

CoopnBuilder can generate graphcal representation of classes or contexts. Just select the "GraphEdit" tab at the bottom of the editor panel, you'll get the following graph (layout by hand):

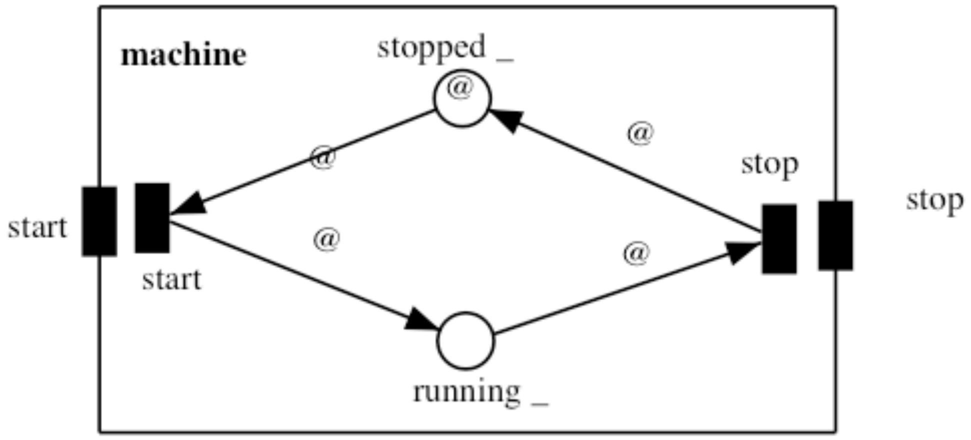

Figure 2.

We've got the same Petri net as Figure 1.

The above Petri net is more like a state machine, later we'll show an example with multiple tokens in a place. Now we'll use Algebraic Abstrait Data Types (ADTs) to represent tokens in Petri net: a more powerful approach to represent the state of a Petri net. The expressiveness of algebraic ADTs makes COOPN as a powerful language to modelize interactions between systems.

We define an ADT "**State**" with finite number of possible values: **stopped** and **running**. We mention this because ADTs may have infinite number of values, like **natural** and **integer**. ADTs can also have operations and theorems, here we just enumerate the possible values using "**Generators**".

```
ADT State;
Interface
    Sort
        state;
    Generators
        stopped : -> state;
        running : -> state;
End State;
```

We have "**stopped**" and "**running**" as **sort** of **state**.

Then we define another machine class using **State** at the place of **BlackToken**, note that **the definition of place and axioms changed but the interface (methods) did not**:

```
Class machine2;

Interface
    Type
        machine2Type;
    Methods
        start;
        stop;
Body
    Use
        State;
    Place
        mState _ : state;
    Initial
        mState stopped;
    Axioms
        start::mState stopped -> mState running;
        stop::mState running -> mState stopped;

End machine2;
```

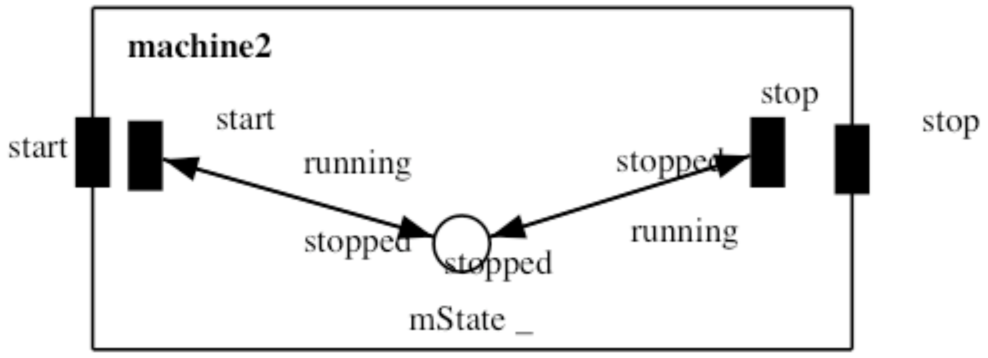

Figure 3.

We have another equivalent Petri net with ADT, comparing with Figure 2., we observe that:

- only one place is need because we use an ADT **State** to represent the state of machine. **machine2** is more compact and it's also easy to understand.
- from the outside of module, we don't see the difference between **machine** and **machine2,** because the interface did not change. We are able to encapsulate a Petri net as an object and seperate its interface from its implementation (intenal behavior).

Ang Chen [http://cui.unige.ch/%7Echena/] @ SMV [http://smv.unige.ch] , 6 April, 2005