Petri **N**et **De**sign **S**tudio

Your mini-project task will be to create a design studio that is working with Petri-Nets. Although you have the freedom to implement your own solution and you are encouraged to do so, there are certain definitions and features that your studio has to follow/provide.

Definition

A Petri net is defined as a triple (P, T, F) where:

- P is a finite set of *places*
- T is a finite set of *transitions* ($P \cap T = \varnothing$)
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation) {for short we will write $f(p \rightarrow t)$ to describe an arc that connect transition t to place p}

Additionally, *marking* $M \in P \rightarrow Z^*$ is a function that assigns a non-negative integer to every place and represents the state of the net (some definitions call this a marked petri net). M(p) denotes the marking of place p. *Inplaces* of a transition (*t) is a set of places where each element of a set is connected to the transition (the place is the source of the arc and the transition is the destination). Conversely, *outplaces* of a transition (t*) is a set of places that are connected to the transition by *arcs* where the places are the destinations and the transition is the source.

The following definitions cover how the petri net progress from one marking to another:

- $t \in T$ is *enabled* if $\forall\ p \in P \mid \exists\ f(p \rightarrow t) \in F\ M(p) > 0$ - for all *inplaces* of the transition (that are connected to the transition via an incoming arc) the amount of tokens at the place is non zero
- *Firing* an enabled transition decreases the amount of tokens on all *inplaces* with one and increases the amount of token in all *outplaces* of the transition by one.

Classifications

We recognize the following features of a petri net:

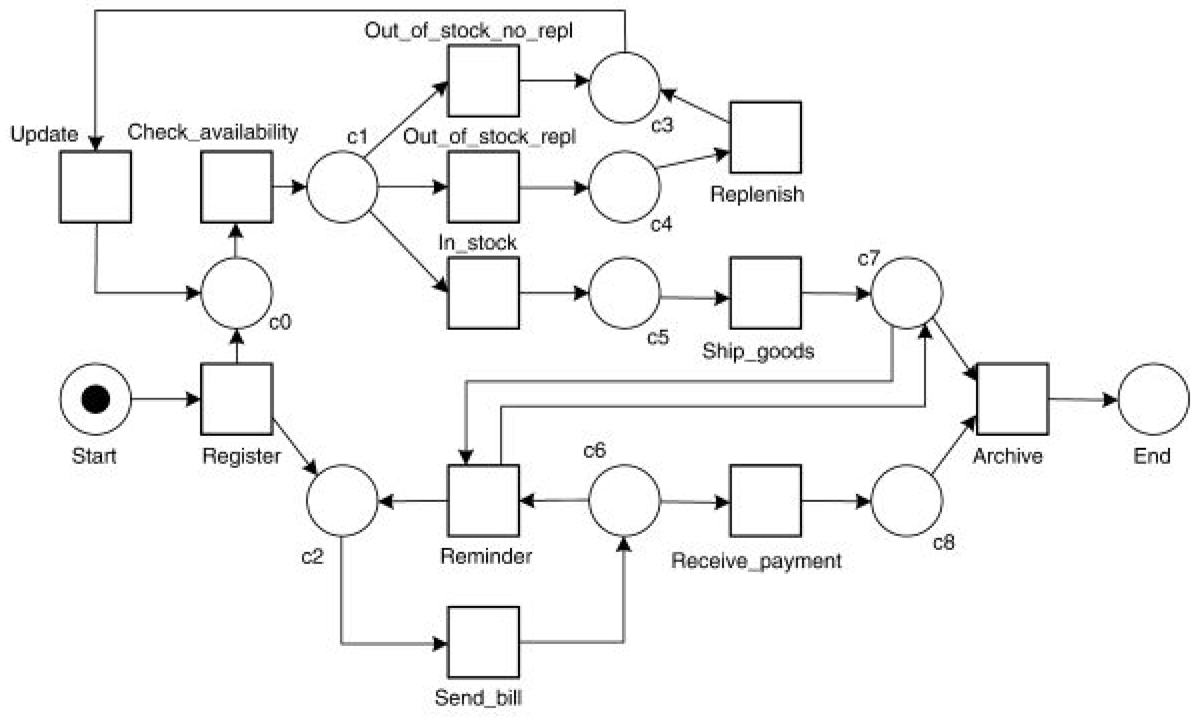
- *Free-choice petri net* - if the intersection of the inplaces sets of two transitions are not empty, then the two transitions should be the same (or in short, each transition has its own unique set if *inplaces*)
- *State machine* - a petri net is a state machine if every transition has exactly one *inplace* and one *outplace*.
- *Marked graph* - a petri net is a marked graph if every place has exactly one out transition and one in transition.
- *Workflow net* - a petri net is a workflow net if it has exactly one source place s where $^*s = \varnothing$, one sink place o where $o^* = \varnothing$, and every $x \in P \cup T$ is on a path from s to o.

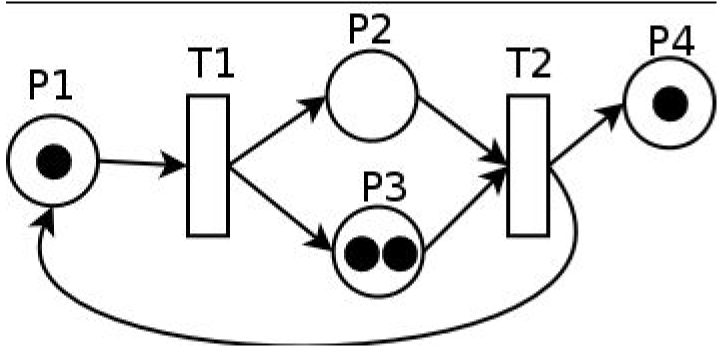
Studio

You are to implement the PeNDeS. To successfully achieve this goal, you have to create a github repository to contain your design studio code. Then, you need to build a project seed containing the Petri Net metamodel that adheres to the rules of the mathematical description (Do not forget that arcs can only connect place to transition or transition to place!). Additionally, you need to make sure that your network is appropriately decorated (see below). Your next step will be to implement a simulation visualizer (see specific requirements later) that allows the modeler to check the behavior of the network he/she just created. Finally embed an interpreter into the toolbar of the visualizer that goes through the classifications and notifies the user whether the given network falls into one category or not.

Decoration

You need to make the model editor to resemble the graphical syntax of petri nets:





- Places are circles
- Transitions are squares (alternatively, the ‘black bar’ decoration is acceptable - that is most common among petri net editors)
- Markings should be highlighted inside each place (black disks up to 12, over that a number is fine)
- Additionally, the decoration of a network should include the initial marking in textual form (M: placeName-marking otherPlaceName-otherMarking … finalPlaceName-finalMarking)

Use of SVG decoration is highly recommended (though you can implement a complete decoration if you want).

Simulator
Your simulator should implement the following features:

- Should visualize the network similarly to the composition
- Additionally, it should differentiate the transitions that are enabled
- Firing should happen once the user clicks on an enabled transition
- Markings should progress according to firings (no animation is required, but would be nice)
- The visualizer should have a 'reset' button on its toolbar that switches the network back to the initial marking
- The state of the simulation should not be reflected in the model
- If your network reaches a deadlock (there is no enabled transition), some visual effect should notify the user (or an actual notification…)

Interpreter
The interpreter that should be tied to a toolbar button can use any implementation to check the individual classifications. Using formula is not necessary though probably makes the checking easier (do not forget that you will need to add the formula code to your repository or have some install instructions so the end user of your domain will be able to install it properly).

Example model
You also have to create some example models. Make sure they have nice descriptions so that beginner users will understand what problems they can solve with your design studio.

Documentation
You finally have to create documentation to your design studio. It can be as simple as a readme.md file in the repository, but it has to clarify the following things for the user:

- What is the domain about
- A few sentence on the typical use-cases of the domain
- How to install the design studio
- How to start modeling once the studio is installed
- Once a network is build, what feature your studio provides and how can the user use those functions

The documentation should roughly be a one pager, but there is not actual upper limit…

!!! Do not be afraid if some elements are not yet clear. Make sure that you start by understanding your domain and by creating the metamodel. By the time you are done with those, we will talk about all other technologies that you might need.