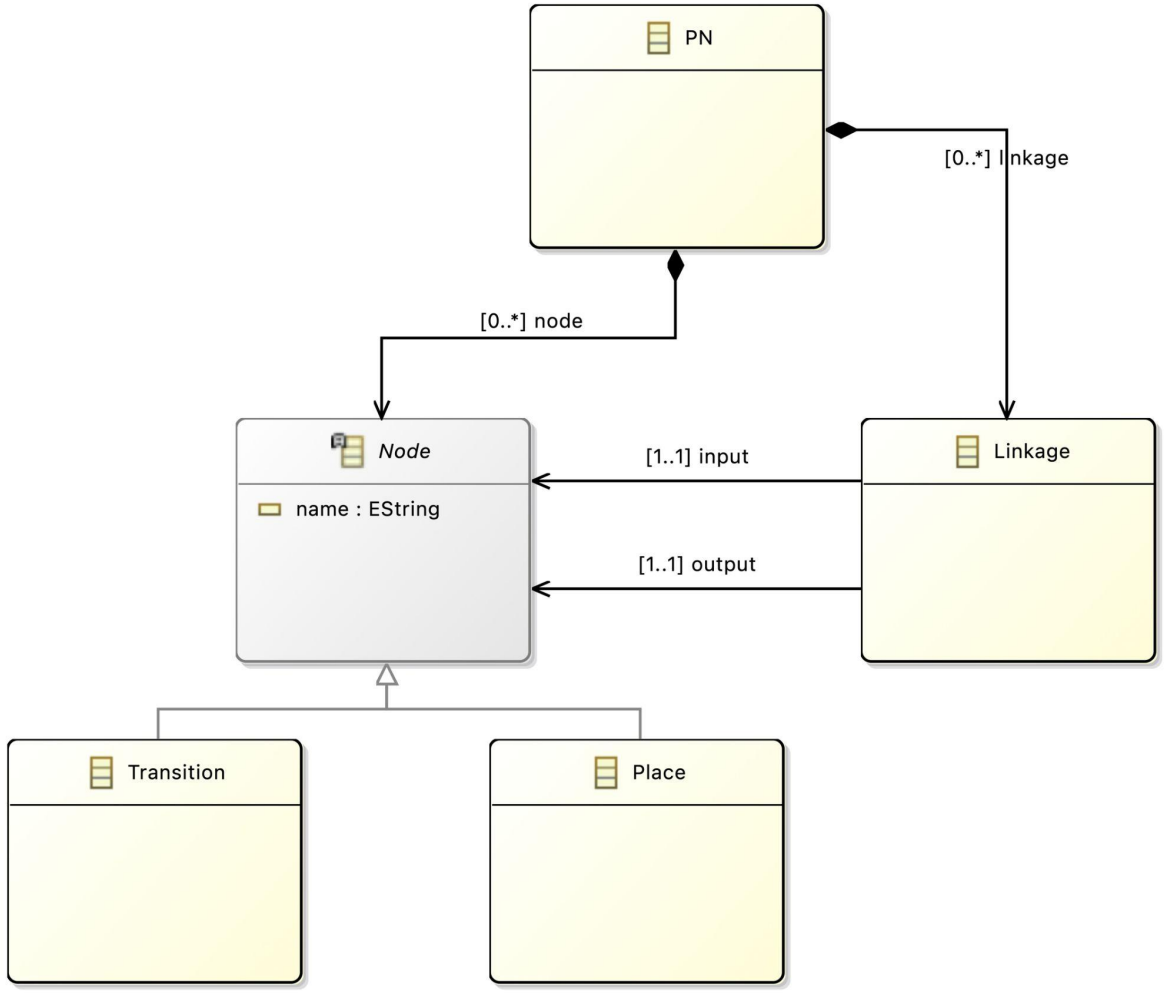
UML Sequence Diagrams And Petri Nets Interoperability Project

Mykola Fedurko

INTRODUCTION

In this part of the project, we are generating code for the Petri nets model to interoperate with a tool. Let's again take a look at our metamodel of a Petri nets:



Picture 1. Petri nets metamodel

Petri nets is itself a pretty minimal modeling language(partly because of its mathematical origins). So its metamodel is self-describing as was stated in the previous paper about the design of the

metamodels.

In this part of our project description, we are aiming at describing the code generation step: state the alternative target tools, generation tools and describe our approach. So above is just a brief reminder of what exactly we are going to generate.

From now on in this paper, the PN stands for Petri nets.

ALTERNATIVES AND JUSTIFICATION OF CHOSEN SOLUTION

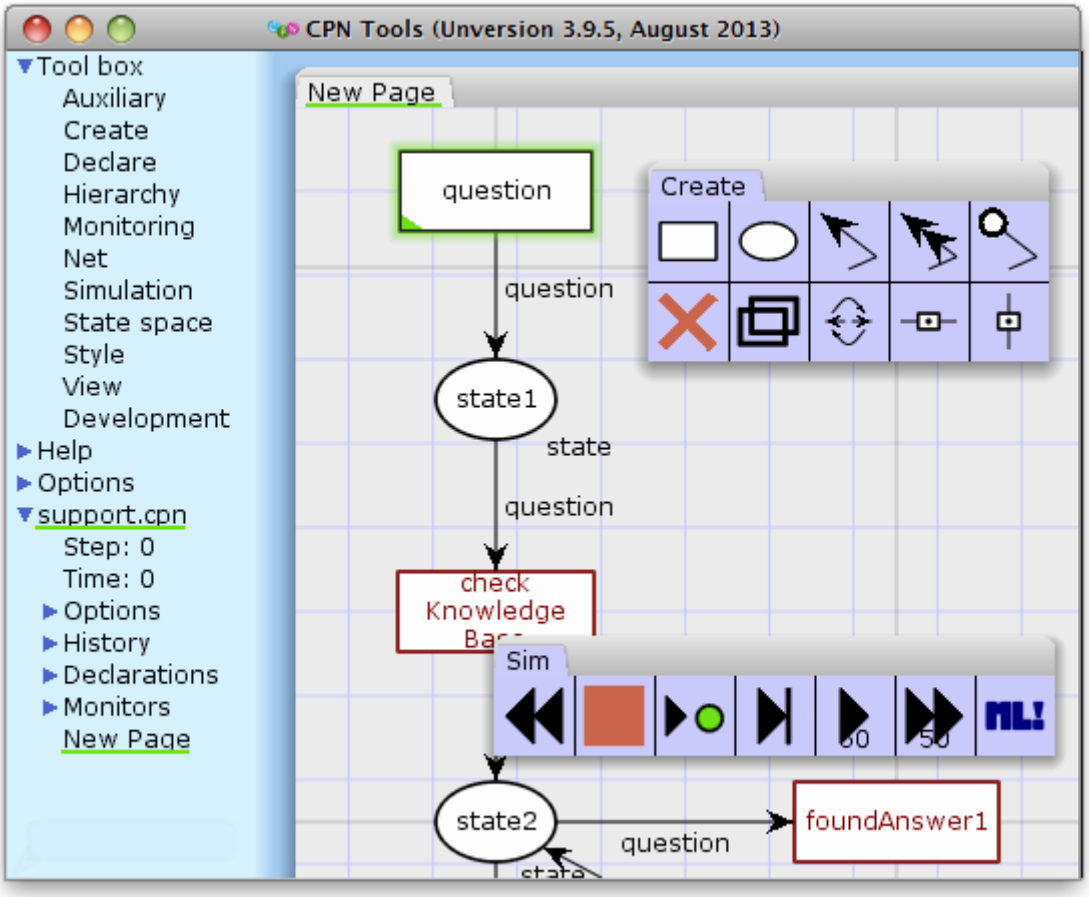
Just going out and googling “petri nets drawing tools” gives you a few interesting results, the most notable of which are:

- CPN Tools
- PIPE2

So our goal is to choose between these two Petri nets modeling tools.

CPN Tools

First of all let's note that CPN is a more popular choice and more well supported one. It stands for Coloured Petri Nets Tools.



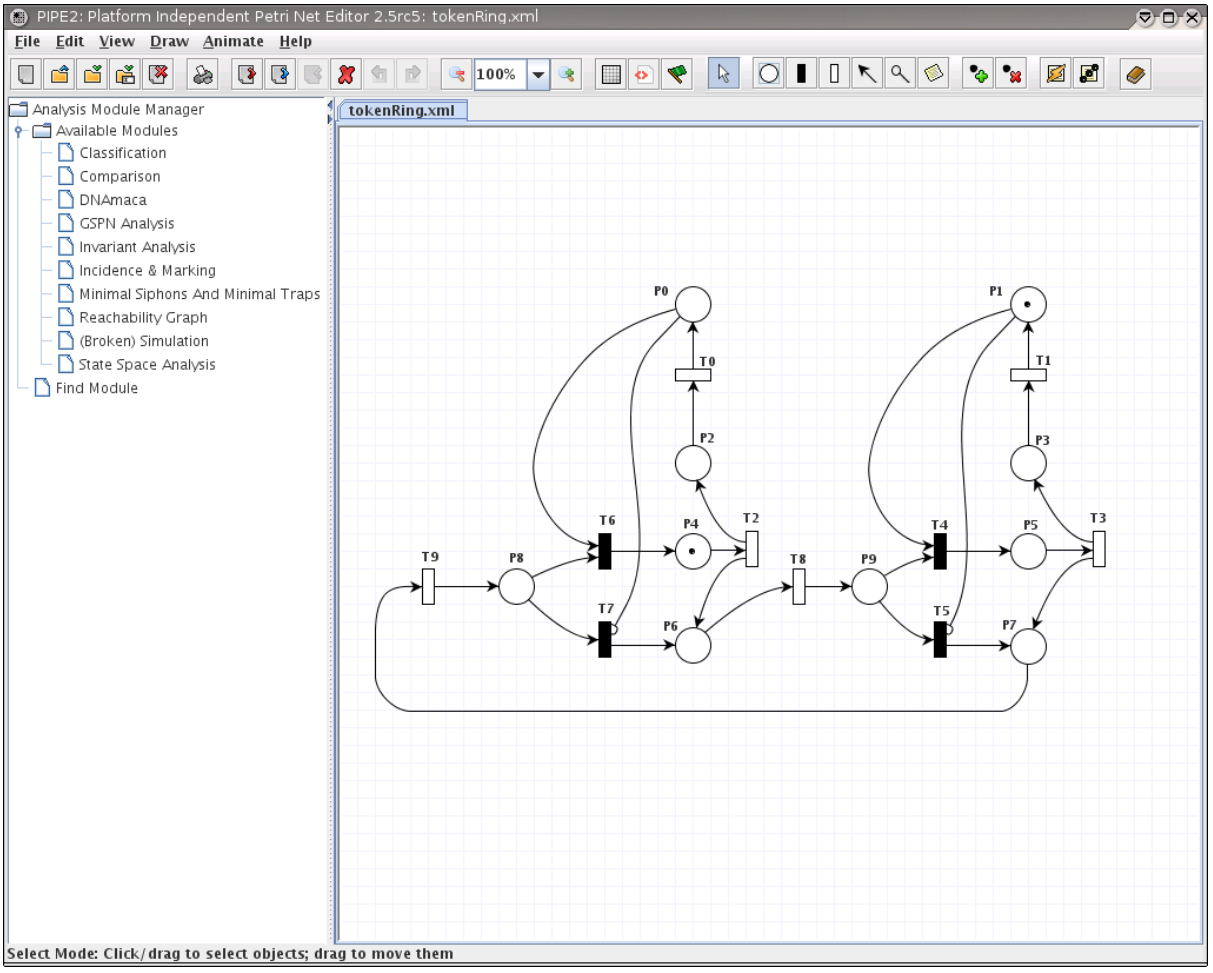
Picture 2. CPN Tools UI

This option basically offers a plethora of options: different arc types, transition and places types. A few modes of speed and lots of different addons to classic

Petri nets. Obviously, deducing from its naming: an option to describe Coloured Petr nets. Also the tool is fully cross platform, with Linux, MacOS and Windows packages available.

PIPE2

PIPE2 stands for Platform Independent Petri net Editor 2. The thing to note right from its name is that this tool is too fully cross platform. The biggest emphasis is done on analysis of classical Petri nets(Available Modules on the left) and on simplicity:



Picture 3. PIPE2 UI

There's only simple Place, Arc and Transition, with Timed transition being the most fancy one. That's it - all the other things are bare minimum enough for covering the behavior of Petr nets.

Decision

So taking a look at two of these tools the second one of them(PIPE2) was decided to be chosen for code generation, with following perks:

- Simplicity in UI
- As a consequence, simplicity in file format. Both of them use xml to describe Petri nets, but the PIPE2 xml file is much shorter than CPN one.
- No burden of Coloured Petr nets, just classical ones to take into account.

Also there's one con to that choice - PIPE2 is less popular and less supported than CPN. But as the purpose of this project is just to generate a proof of concept working solution it's a suitable option.

Choice of generation tool

Basically, we had not had many tools to choose from due to the novicity of MDE to authors. First of the options is Acceleo - a modern tool for generation of code from models.

And considering other options familiar to authors would mean going with the Java: writing classes, implementing interfaces and so on. As much as we like Java we really like DSLs with Acceleo being the one for generating code.

So as always it's better to use dedicated DSL for its concise syntax and powerful abstractions(instead of building them on our own).

SOLUTION DESIGN

The solution design depends on file format of PIPE2:

```
<place id="START_0">
    <graphics>
        <position x="170.0" y="255.0"/>
    </graphics>
    <name>
        <value>START_0</value>
        <graphics>
            <offset x="0.0" y="0.0"/>
        </graphics>
    </name>
    <initialMarking>
        <value>Default,0</value>
        <graphics>
            <offset x="0.0" y="0.0"/>
        </graphics>
    </initialMarking>
    <capacity>
        <value>0</value>
    </capacity>
</place>
```

Picture 4. Place in PIPE2

The “movable” parts are: graphics->position, name->value. All of the others are static values that won’t change.

```
<transition id="T6">
    <graphics>
        <position x="380.0" y="465.0"/>
    </graphics>
    <name>
        <value>T6</value>
        <graphics>
            <offset x="-5.0" y="35.0"/>
        </graphics>
    </name>
    <orientation>
        <value>0</value>
    </orientation>
    <rate>
        <value>1.0</value>
    </rate>
    <timed>
        <value>false</value>
    </timed>
    <infiniteServer>
        <value>false</value>
    </infiniteServer>
    <priority>
        <value>1</value>
    </priority>
</transition>
```

Picture 5. Transition in PIPE2

The “movable” parts are: graphics->position, name->value. All of the others are static values that won’t change.

```
<arc id="START_0 to T6" source="START_0" target="T6">
    <graphics/>
    <inscription>
        <value>Default,1</value>
        <graphics/>
    </inscription>
    <tagged>
        <value>false</value>
    </tagged>
    <arcpath id="4" x="136" y="237" curvePoint="false"/>
    <arcpath id="5" x="252" y="223" curvePoint="false"/>
    <type value="normal"/>
</arc>
```

Picture 6. Arc in PIPE2

The “movable” parts are: id, source, target in arcpath.id(that should be unique). All of the others are static values that won’t change.

And also there’s of course header, which is static**:**

```
<?xml version="1.0" encoding="ISO-8859-1"?><pnml>
    <net id="Net-One" type="P/T net">
        <token id="Default" enabled="true" red="0" green="0" blue="0"/>
```

Picture 7. Header in PIPE2

So first of all we put a static header into our target file. Then we are iterating over each of the nodes in PN and getting its name assigned to place/transition id and name->value.

Also there's a problem of shaping the visual position of each node. We decided to put it into a grid of width 5 by generating x and y position like that(where nodei is index of node in a PN.nodes sequence):

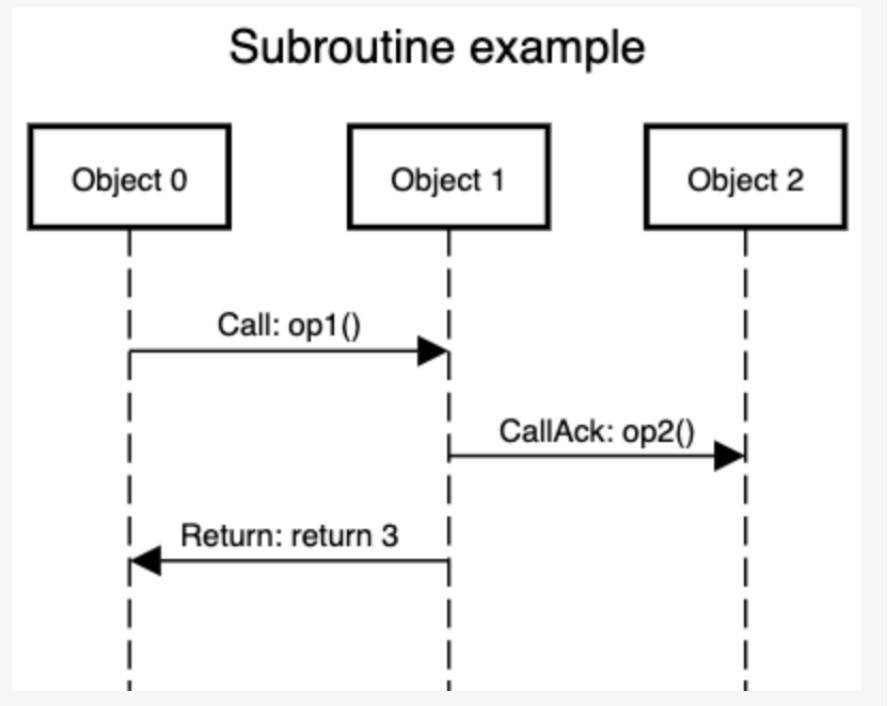
```
<transition id="[n.name/]">
    <graphics>
        <position x="[100+70*(nodei.mod(5))/].0" y="[255 + (nodei/5).floor()*70/].0"/>
    </graphics>
```

Picture 8. Generating node position in PIPE2

And lastly we have Arcs generation. With arc id assigned to "linkage.input.name to linkage.output.name" and source assigned to "linkage.input.name" and target assigned to "linkage.output.name". Also we didn't forget about unique arcpath.id and decided to just get it generated by a counter variable.

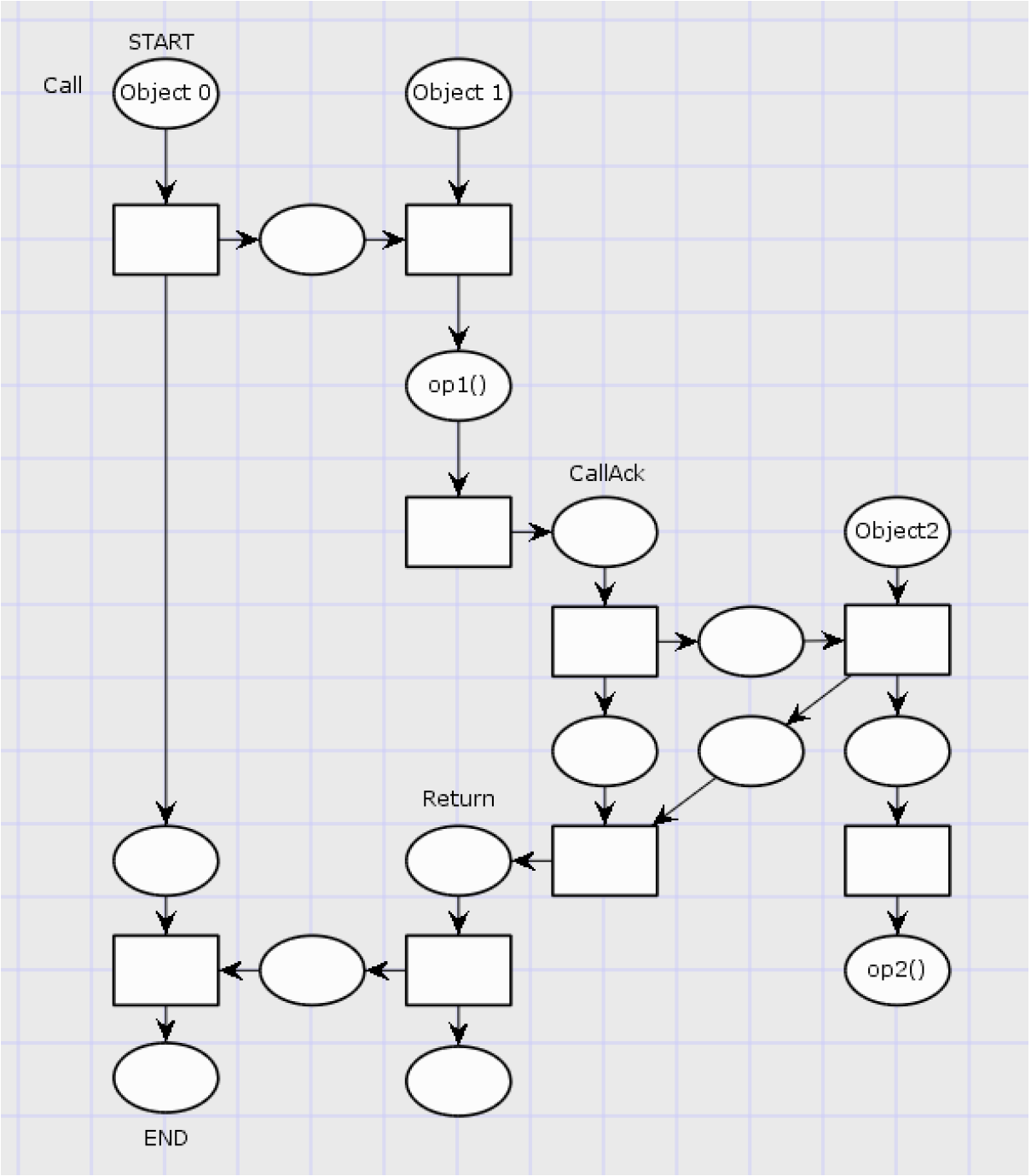
TESTS AND EVALUATION

From previous paper we were transforming the following Sequence diagram:



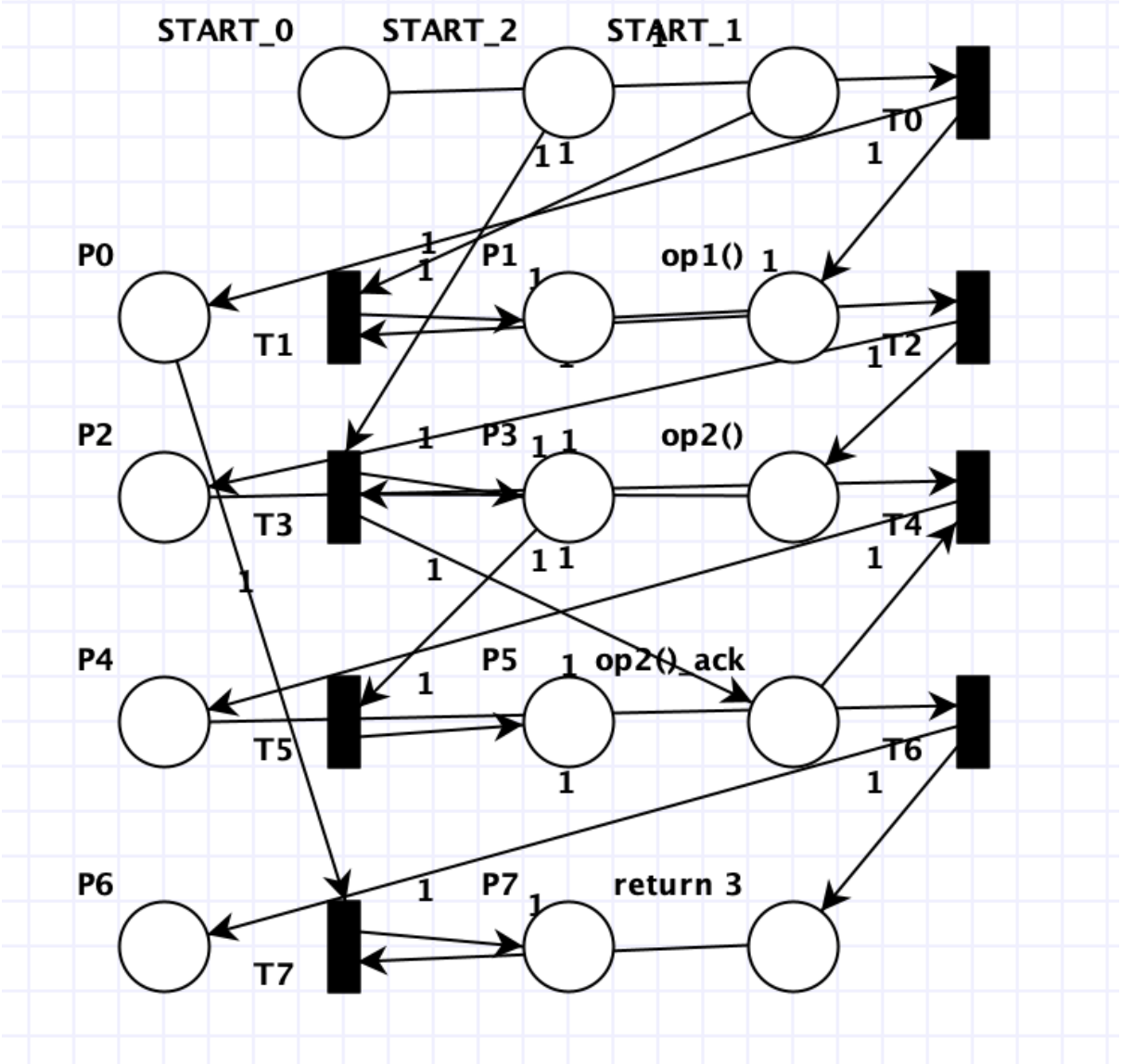
Picture 9. Subroutine sequence diagram

And it was transformed into the following Petri net diagram:



Picture 10. Subroutine Petri net diagram

So regarding our solution, this Petri net model was transformed into a following PIPE2 model:



Picture 11. PIPE2 Subroutine model

So while this one is not so pretty anymore, at least it's readable with all the nodes being in different places and only arcs crossing each other. So it's a good tradeoff. Maybe in future versions of this tool we could do better by providing more concise visual representation. But it seems that it may require to include some of AI to solve this problem, because it's a matter of intuitiveness, which is something that machines is not very good at.

REFLECTION

During the makings of this project lots of new things were tackled. We certainly dug into Eclipse, Ecore, UML, QVT, Acceleo, CPN, PIPE2, Petri nets and other tools regarding MDE. Not everything was easy: as such there's a huge lack of documentation for these tools and I think this is something that could be improved for them. Also the eclipse environment may be buggy sometimes, because open source is not very good at UI or UX. But nevertheless the product was getting its job done and most of the time was working predictable.

Regarding the work itself we were concerned with transforming Sequence diagrams into Petri nets to interoperate with the tool PIPE2. We chose a limited version of SD and a classical minimal version of PN, and PIPE2 was chosen also because of the simplicity of its working. So the major flow of this work is that it's not connected to a practical world. In a future version of it we should do better at covering full specifications of the aforementioned tools. And perhaps even create a pipeline for generation of CPN compatible file format so that more users could enjoy our solution.