

UML Sequence Diagrams And Petri Nets Interoperability Project

Mykola Fedurko

PURPOSE OF THE PROJECT

The purpose of the project is to create a transformation mechanism from UML Sequence diagrams to Petri nets to enable further analysis of Sequence diagrams with a powerful mathematical tooling of Petri nets. In this project we assume only the mere basics of the functioning of these.

For example in Sequence diagrams we do not include different kinds of objects(Actor, Entity, Boundary, Control) - we only have standard Object type. So we model sequence diagrams as a sequence of messages between objects. We only have “call”, “return” and “call with acknowledgement” message types in this project to serve as a proof of concept of control flow in an ordinary programming language. Also we don't model any comments on sequence diagrams(from now on just SD).

And in Petri nets we only model simplistic structure: “place”, “transition” and “linkage”. “place” and “transition” are all types of “node” in this project. And it's all working under the assumption that no linkage is possible between the nodes of the same type.

ALTERNATIVES AND JUSTIFICATION OF CHOSEN SOLUTION

The alternative to the current approach may be considered a more complex, closer to the original definition of these tools design. But the point of these work was not to mimic all the features of the tools used, but to create a proof of concept work for further analysis.

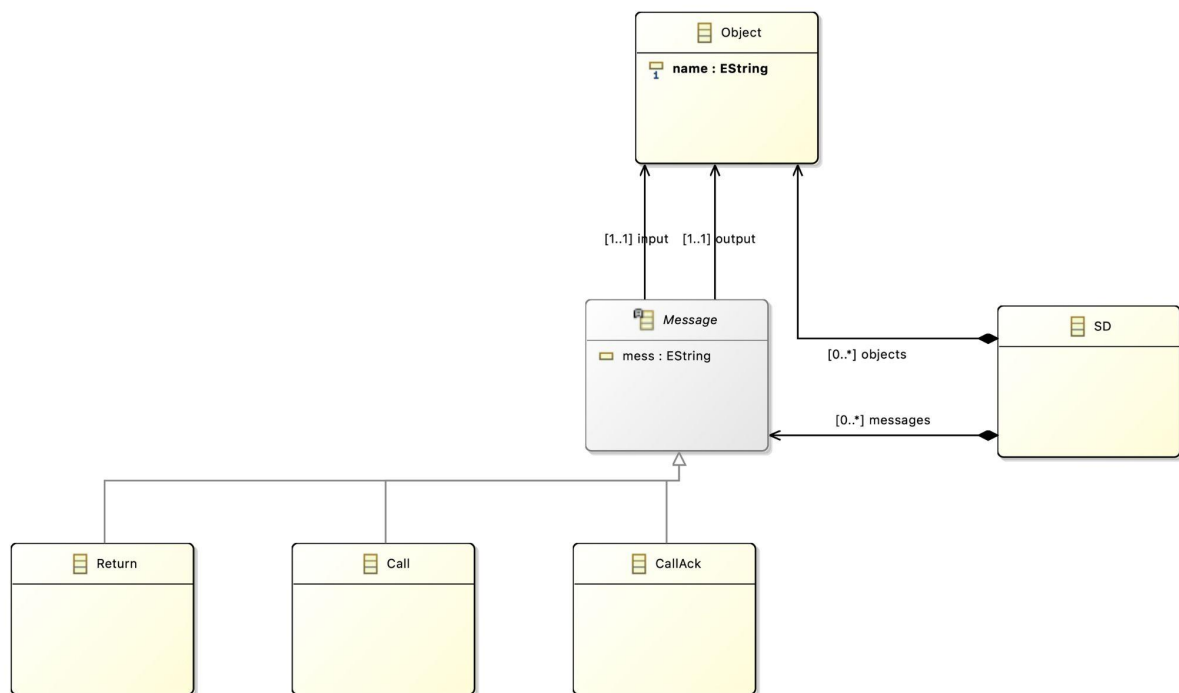
There's a [UML sequence diagram metamodel](#) widely available on the internet, and it provides a standardized approach to modeling UML. And a perhaps not less strictly standardized [Petri net metamodel](#) from PNML. But as was described above it's not quite the match for the purposes of this work.

Nevertheless it's perfectly possible to create new model transformations that would help to transform models of “standardized metamodels”(the ones from the links above) into the models of a metamodels presented in this work. It's just that right now it's not the aim of this project.

SOLUTION DESIGN

To give reader an overview of the project, let's present UML diagrams right away:

Sequence diagrams metamodel



Picture 1. Sequence diagrams metamodel

The root of a diagram is an SD object - that is just a container for all the Messages and Objects in a diagram. There are two main objects in this diagram: Message and Object - the same corresponding notions of any UML sequence diagram. Every Message object has exactly one input and output Object as it's source and destination.

To find out the complete specification of messaging one can just take a look at a messages sequence. With these and Object instances you can create any sequence diagram. There's three types of messages:

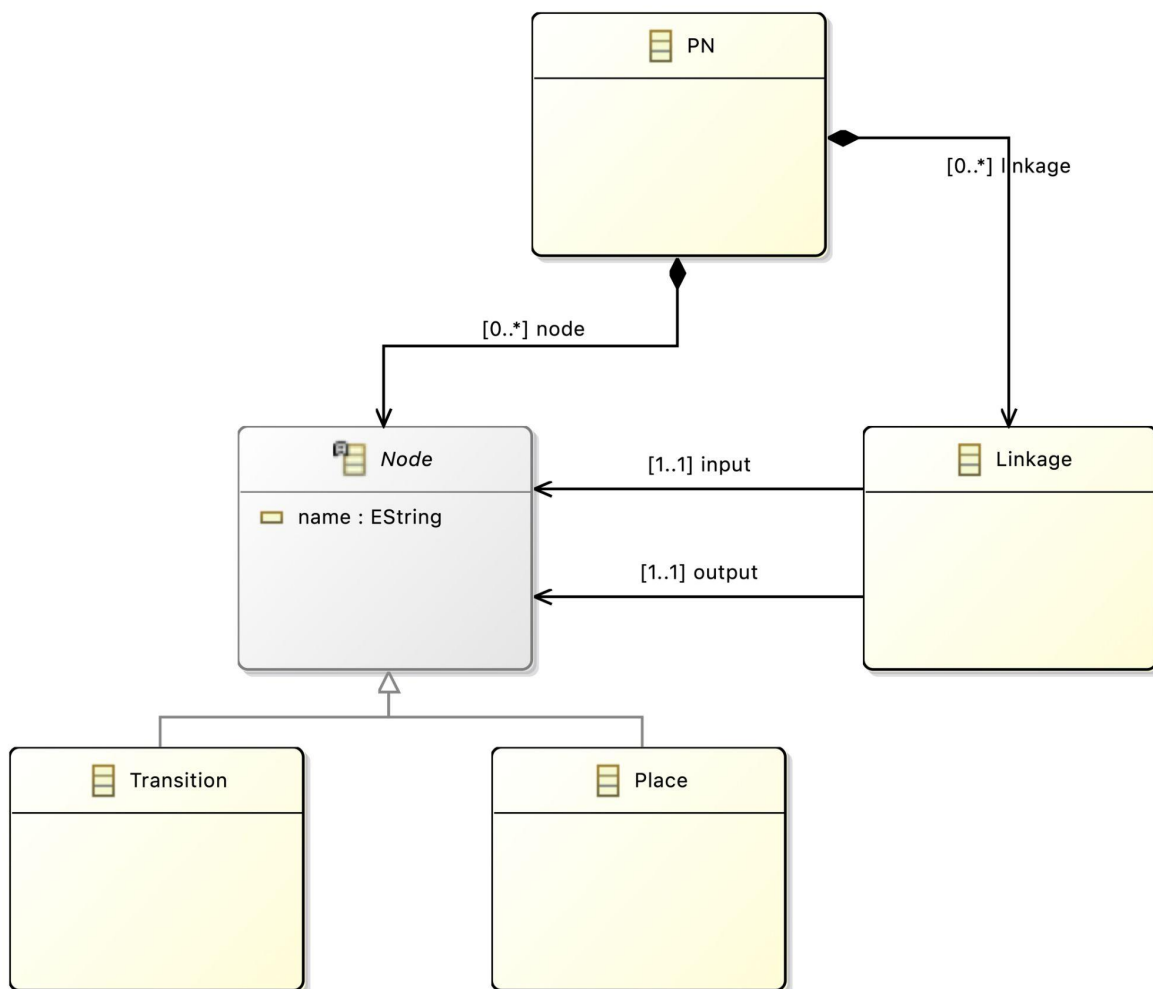
- Call - just a call from one object to another(Just async call, not caring for the result)

- Return - return(may only be placed after a Call to some object occurred - meaning Sync communication)
- CallAck - loosely sync communication(we only want to know if the message is received, not caring for the result).

So basically there's three types of communication:

- Async call(corresponding to Call message in SD)
- Sync call(corresponding to Call message from ObjectA to ObjectB and Return message from ObjectB to ObjectA)
- Loosely sync call(corresponding to a CallAck message in SD)

Petri nets metamodel



Picture 2. Petri nets metamodel

Petri net metamodel root node is a PN object. It contains all of the Node and Linkage objects. Linkage is a connection between two Nodes: input and output.

There's two subtypes of Node:

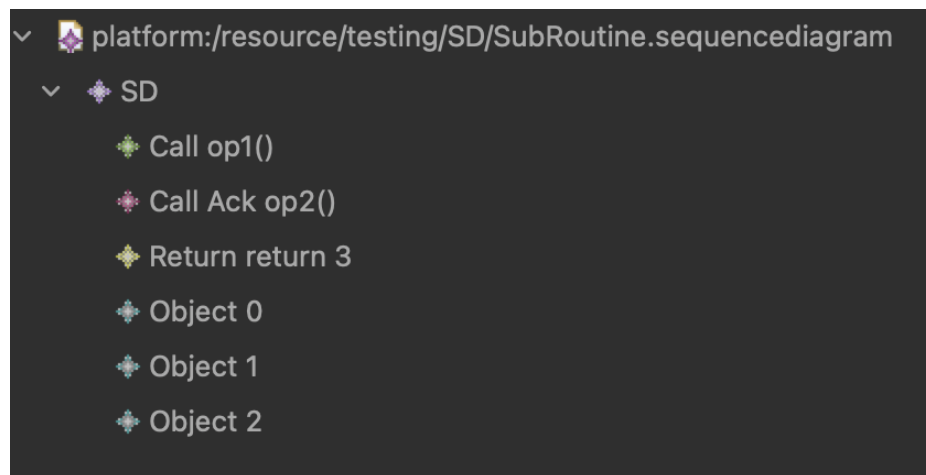
- Place - a place in Petri net terminology
- Transition - a transition in Petri net terminology

There's also an invariant in OCL saying that there can't be a linkage between two Nodes of the same type. Generally it is very simple and easy to grasp once you know bare minimum about Petri nets(which we are assuming the reader does).

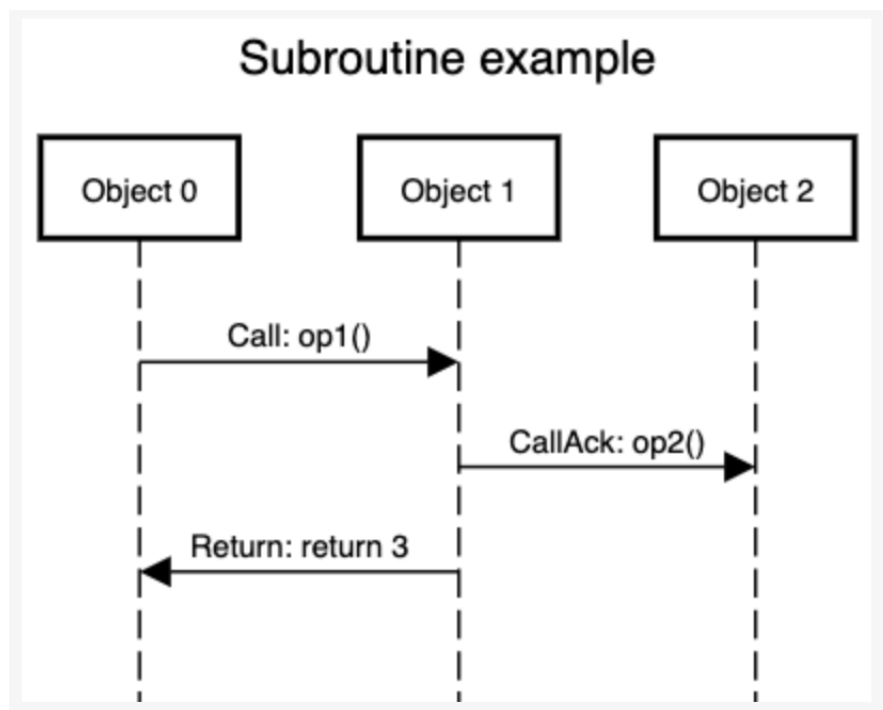
TESTS AND EVALUATION

Sequence diagrams

Here's the subroutine example. On picture 3 you can see our model and on picture 4 you can see the corresponding diagram.



Picture 3. SubRoutine model



Picture 4. SubRoutine sequence diagram

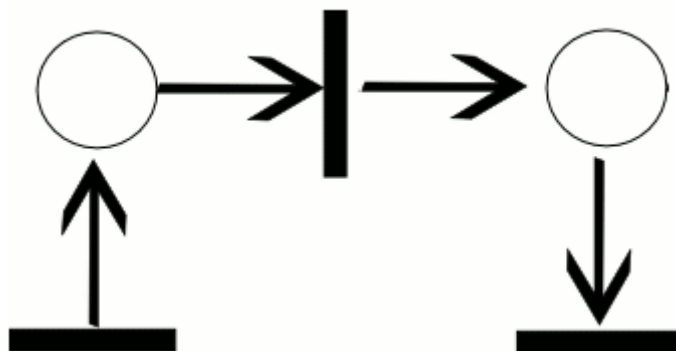
The biggest limitation as for now is that we don't tie Returns to any particular call via references, we just rely on the user to provide appropriate data.

Petri nets

Here's a simple example. On picture 5 you can see our model and on picture 6 you can see the corresponding diagram.

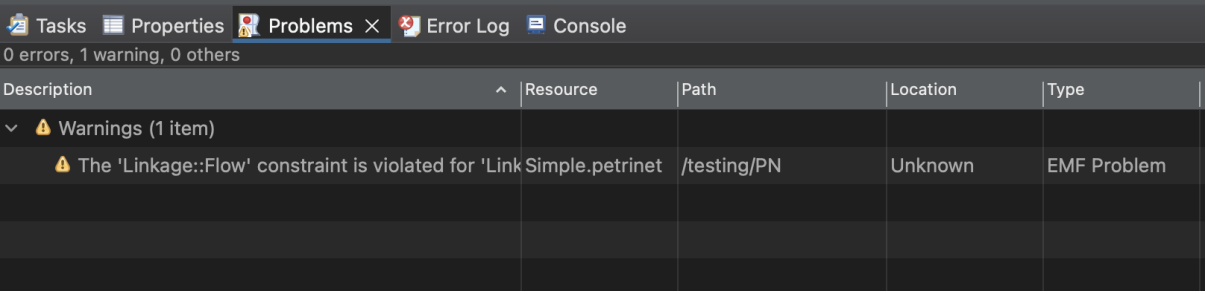


Picture 5. Simple Petri net model



Picture 6. Simple Petri net diagram

Although if we switch any of the Linkages to Transition-Transition or Place-Place type, then the editor will give us an error:



The screenshot shows the 'Problems' panel in an IDE. The panel has tabs for 'Tasks', 'Properties', 'Problems' (selected), 'Error Log', and 'Console'. Below the tabs, it says '0 errors, 1 warning, 0 others'. The main area is a table with the following columns: 'Description', 'Resource', 'Path', 'Location', and 'Type'. There is one warning item expanded, showing a yellow warning icon and the text: 'The 'Linkage::Flow' constraint is violated for 'Link Simple.petrinet'. The table data is as follows:

| Description | Resource | Path | Location | Type |
|--|-----------------|-------------|----------|-------------|
| ⚠ Warnings (1 item) | | | | |
| ⚠ The 'Linkage::Flow' constraint is violated for 'Link Simple.petrinet | Simple.petrinet | /testing/PN | Unknown | EMF Problem |
| | | | | |
| | | | | |

Picture 7. Wrong Linkage error