Integration Test using Open-Timaeus-Net

Open-Timaeus-Net purpose

In the context of critical embedded systems, verification of temporal and functional constraints in the worst-case is an essential property, not only to ensure the proper functioning of the system in its environment, but also and specially to guarantee strict certification requirements, particularly for avionics and space.

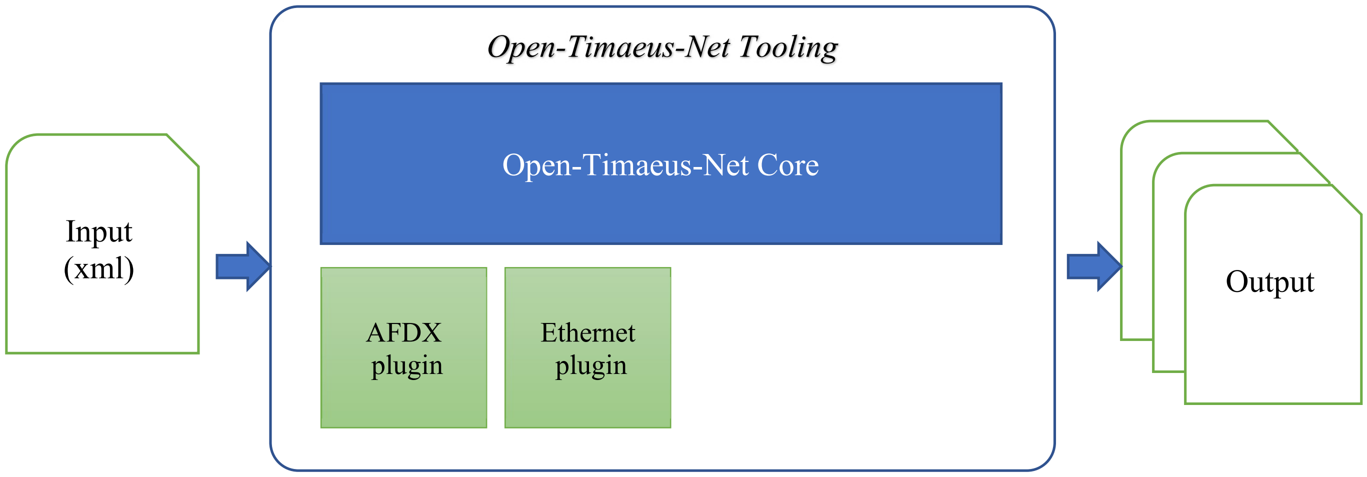
The opportunity to make this worst-case performance analysis since the early design phases will allow designers to make important decisions concerning the system parameters tuning and dimensioning, to avoid wasting time and costs in detailed and erroneous implementation.

The tool Open-Timaeus-Net provides a first answer to this design challenge. The first applications show that Open-Timaeus-Net reduces development time and costs, while meeting the system requirements since the early design phases.

Open-Timaeus-Net architecture & workflow

Open-Timaeus-Net is a standalone tool developed as a RCP Eclipse application. Such Eclipse RCP application consists of of plugins: some of them are part of the "Core" of Open-Timaeus-Net; some others are not mandatory and are extensions plugins: for instance, the AFDX technology and the related analysis are implemented inside Open-Timaeus-Net based on an additional plugin.

As shown in the following figure, the input of Open-Timaeus-Net is usually an xml file describing the network and the traffic. The output is a set of performance metrics, such as end-to-end delays, backlogs, jitters and link loads.



Example of Input :

```
<?xml version="1.0" encoding="UTF-8"?>
<elements>
  <network name="Z_ESE" overhead="67" shortest-path-policy="DIJKSTRA"
  technology="AFDX" transmission-capacity="100Mbps" x-type="FULL"/>
  <station name="AFDX Station 5" service-policy="FIRST_IN_FIRST_OUT"
  transmission-capacity="100Mbps" x="96.0" y="65.0"/>
  <station name="AFDX Station 6" service-policy="FIRST_IN_FIRST_OUT"
  transmission-capacity="100Mbps" x="412.0" y="66.0"/>
  <switch name="AFDX Switch 1" service-policy="FIRST_IN_FIRST_OUT"
  tech-latency="60" transmission-capacity="100Mbps" x="252.0" y="66.0"/>
  <link from="AFDX Station 5" fromPort="0" name="AFDX Edge 3" to="AFDX Switch 1"
   toPort="0" transmission-capacity="100Mbps"/>
  <link from="AFDX Switch 1" fromPort="1" name="AFDX Edge 4" to="AFDX Station 6"
  toPort="0" transmission-capacity="100Mbps"/>
  <flow deadline="1" jitter="0" max-payload="1000" min-payload="1000"
  name="AFDX Flow 1" period="1" priority="Low" source="AFDX Station 5">
    <target name="AFDX Station 6">
      <path node="AFDX Switch 1"/>
      <path node="AFDX Station 6"/>
    </target>
  </flow>
</elements>
```

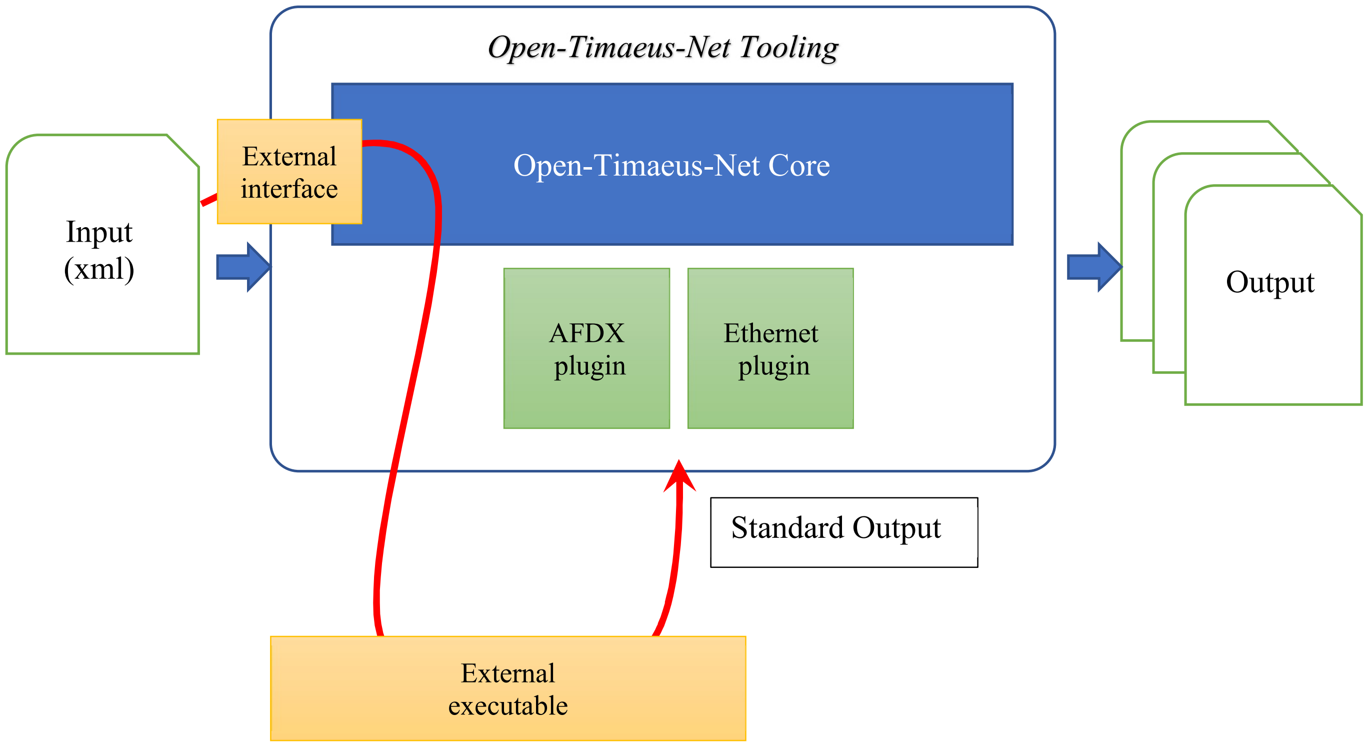
Example of Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<results>
  <delays>
    <flow name="AFDX Flow1 ">
    <target name=" AFDX Station 6" value="45"/>
    </flow>
  </delays>
  ...
</results>
```

Complete description of input and output formats are described in the Appendix **A.**

Open-Timaeus-Net extension with external executable

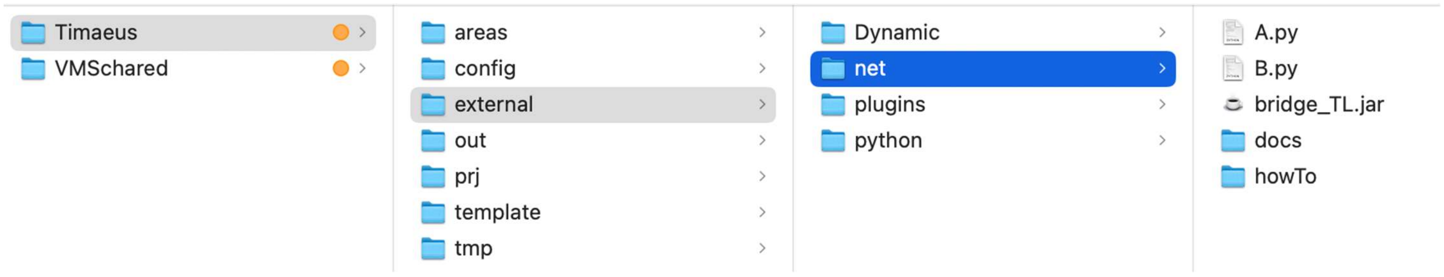
As shown in the following figure, it's also possible to extend Open-Timaeus-Net with external executable. Such an executable has to take as input the specified xml input (the xml file), and produce output results (end-to-end delays, backlogs, jitters, link usage, etc.) according to the specified output.



For this integration phase, the external executables will be your **Python file.**

Extension procedure

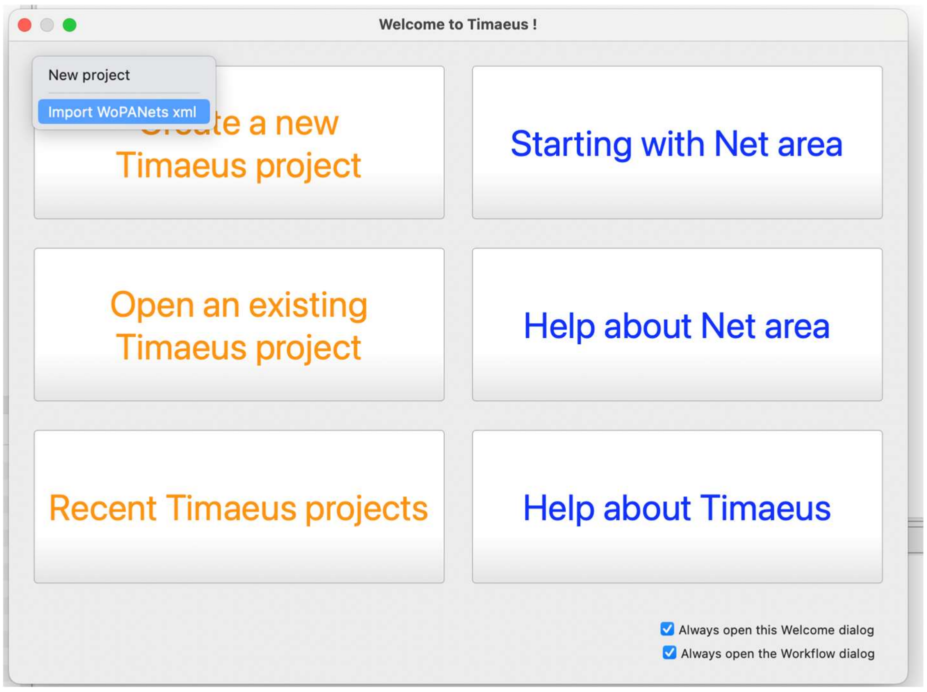
1. Install your Python file in the Open-Timaeus-Net[1] extension folder:



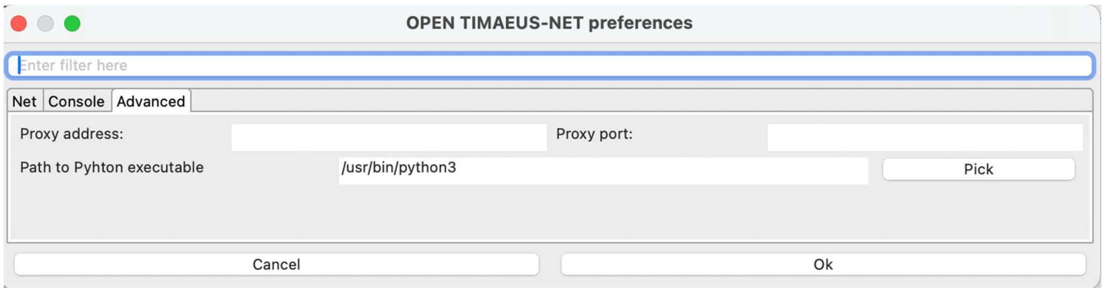
In my case, 2 extension files have been created ("A.py" and "B.py").

2. Import an XML file inside Open-Timaeus-Net:

[1] The Open-Timaeus-Net folder is located in your **personal** document folder.

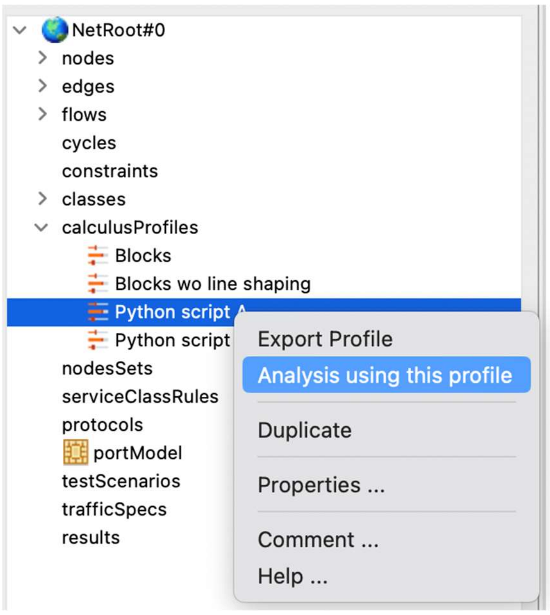


3. Inside the “Preferences” (Advanced Tab), check the path to the python3 executable (and change it clicking on “Pick” if this path is not correct)

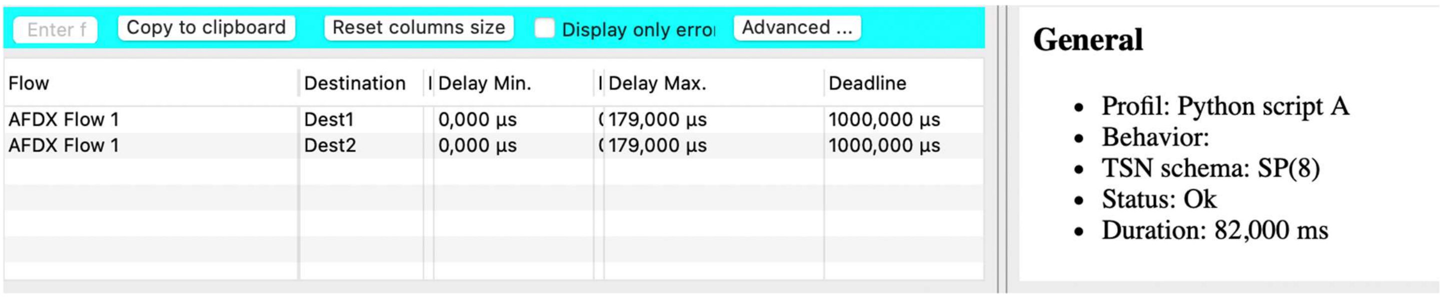


Proxy address:
Proxy port:
Path to Pyhton executable /usr/bin/python3

4. Then inside Open-Timaeus-Net, select your calculus profile (according to your script) and launch the analysis:



When the computation is done, some windows with the results are shown:



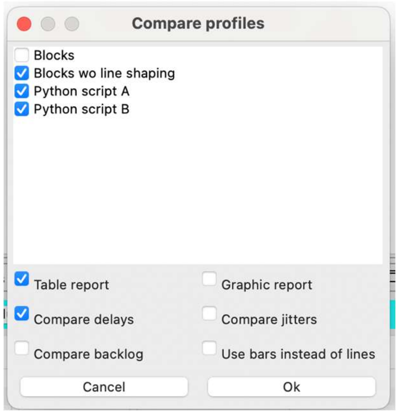
Flow	Destination	Delay Min.	Delay Max.	Deadline
AFDX Flow 1	Dest1	0,000 µs	179,000 µs	1000,000 µs
AFDX Flow 1	Dest2	0,000 µs	179,000 µs	1000,000 µs

General

- Profil: Python script A
- Behavior:
- TSN schema: SP(8)
- Status: Ok
- Duration: 82,000 ms

5. The comparison of several engines is possible as follows: Tools/Advanced Analysis/Compare Profiles:

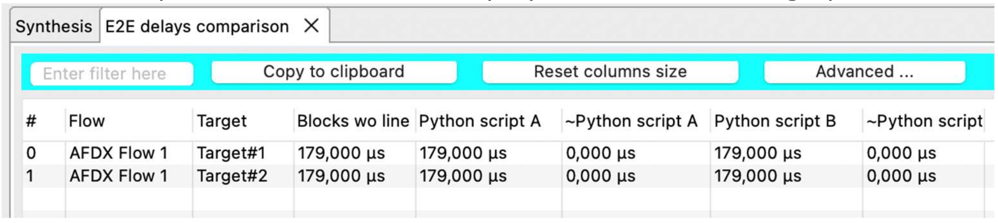
Here I select 3 engines to be compared, and I will compare only the delays.



- [ ] Blocks
- [x] Blocks wo line shaping
- [x] Python script A
- [x] Python script B

- [x] Table report
- [ ] Graphic report
- [x] Compare delays
- [ ] Compare jitters
- [ ] Compare backlog
- [ ] Use bars instead of lines

Comparison results can be displayed inside tables or graphics.



#	Flow	Target	Blocks wo line	Python script A	~Python script A	Python script B	~Python script
0	AFDX Flow 1	Target#1	179,000 µs	179,000 µs	0,000 µs	179,000 µs	0,000 µs
1	AFDX Flow 1	Target#2	179,000 µs	179,000 µs	0,000 µs	179,000 µs	0,000 µs