Parallel DEVS
An Introduction Using PythonPDEVS

Yentl Van Tendeloo, Hans Vangheluwe
Introduction


Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator

A. C.-H. Chow

IBM Corporation, P.O. Box 180124, Austin, Texas 78718.
Tel: 512-838-8069 FAX: 512-250-2346 E-mail: alexc@austin.ibm.com

We present a parallel, hierarchical, modular Discrete Event System Specification (P-DEVS) modeling formalism which provides a modeler with both conceptual and parallel execution benefits. The parallel formalism distinguishes between transition collisions and ordinary external events in the external transition function of DEVS models. Such separation enables us to extend the modeling capability of the collisions. The formalism also does away with the necessity for tie-breaking of simultaneously scheduled events, as embodied in the select function. We next present a design for the parallel simulation procedures needed to prove the formalism’s soundness and to serve as a reference for implementation. We then discuss a prototype implementation that affords a high degree of flexibility by mechanizing the “closure under coupling” property of the Parallel DEVS formalism and the object-oriented characteristics.

Keywords: System specification, discrete event simulation, DEVS, object-oriented modeling and simulation, distributed simulation, modeling methodology

Abstract Simulator for the Parallel DEVS Formalism

Alex ChungHen Chow

Object Technology Products

IBM Corp.
Austin, TX 78758
alexc@austin.ibm.com

Bernard P. Zeigler

Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721
zeigler@ece.arizona.edu
dhkim@ece.arizona.edu

Alex ChungHen Chow, Bernard P. Zeigler and Doo Hwan Kim.

Abstract simulator for the parallel DEVS formalism.
An overview of PythonPDEVS

Yentl Van Tendeloo¹ and Hans Vangheluwe¹,²
¹ University of Antwerp, Belgium
² McGill University, Canada

Yentl.VanTendeloo@uantwerpen.be
Hans.Vangheluwe@uantwerpen.be

An evaluation of DEVS simulation tools

Yentl Van Tendeloo¹ and Hans Vangheluwe¹,²,³

Abstract
DEVS is a popular formalism for modeling complex dynamic systems using a discrete-event abstraction. Owing to its popularity, and the simplicity of the simulation kernel, a number of tools have been constructed by academia and industry. However, each of these tools has distinct design goals and a specific programming language implementation. Consequently, each supports a specific set of formalisms, combined with a specific set of features. Performance differs significantly between different tools. We provide an overview of the current state of eight different DEVS simulation tools: ADEVS, Co+++, DEVS-Suite, MS4, Me, PowerDEVS, PythonPDEVS, VLE, and X-S-Y. We compare supported formalisms, compliance, features, and performance. This paper aims to help modelers in deciding which tools to use to solve their specific problems. It further aims to help tool builders, by showing the aspects of their tools that could be extended in future tool versions.

Yentl Van Tendeloo and Hans Vangheluwe.

An Overview of PythonPDEVS.

Yentl Van Tendeloo and Hans Vangheluwe.

An Evaluation of DEVS Simulation Tools.
Our presentation uses initialized DEVS models, which contain an initial state. The initial state was left implicit in the original DEVS specification.
Sequential Discrete Event Language

- Meijin++
- GPSS
- SimScript
- ...

= modular simulation assembly language
Vangheluwe, Hans. DEVS as a common denominator for multi-formalism hybrid systems modelling.
finite number of non-\( \phi \) events in a finite time interval
Experimentation
Simulation

Model

Solver

Simulator

\[ delay_{red} = 60 \text{s} \]
\[ delay_{yellow} = 3 \text{s} \]
\[ delay_{green} = 57 \text{s} \]
\[ \text{cond}_{termination} = (t_{sim} \geq t_{end}) \]
\[ t_{end} = 24 \text{h} \]

Trace
from pypdevs.simulator import Simulator
from mymodel import MyModel

model = MyModel()
simulator = Simulator(model)
simulator.setVerbose()
simulator.simulate()
INITIAL CONDITIONS in model <trafficSystem.trafficLight>
Initial State: red
Next scheduled internal transition at time 58.50

INITIAL CONDITIONS in model <trafficSystem.policeman>
Initial State: idle
Next scheduled internal transition at time 200.00

INTERNAL TRANSITION in model <trafficSystem.trafficLight>
New State: green
Output Port Configuration:
port <OBSERVED>:
grey
Next scheduled internal transition at time 108.50

INTERNAL TRANSITION in model <trafficSystem.trafficLight>
New State: yellow
Output Port Configuration:
port <OBSERVED>:
yellow
Next scheduled internal transition at time 118.50
Atomic Models
Modelling Discrete Event Behaviour

Finite State Automaton

delay_{red}

red

toRed

delay_{yellow}

green

toYellow

delay_{green}

eyellow

toGreen

e = 0s

Timed Event Scheduling Graph

toRed

red

delay_{red}

yellow

delay_{yellow}

toYellow

green

delay_{green}

toGreen

e = 0s
$\mathcal{S}$: set of sequential states
$\mathcal{S} = \{\text{red, yellow, green}\}$

$\delta_{\text{int}}: \mathcal{S} \rightarrow \mathcal{S}$
$\delta_{\text{int}} = \{\text{red} \rightarrow \text{green, green} \rightarrow \text{yellow, yellow} \rightarrow \text{red}\}$

$\tau_{a}: \mathcal{S} \rightarrow \mathbb{R}_{0,+\infty}$
$\tau_{a} = \{\text{red} \rightarrow \text{delay}_{\text{red}}, \text{green} \rightarrow \text{delay}_{\text{green}}, \text{yellow} \rightarrow \text{delay}_{\text{yellow}}\}$

$\mathcal{M} = \langle \mathcal{S}, \delta_{\text{int}}, \tau_{a} \rangle$

Autonomous (no input)
Time Advance: corner cases

\[ ta : S \rightarrow \mathbb{R}_{0,+\infty} \]

- \( ta(s_i) = 0 \)  
  transient states

- \( ta(s_i) = +\infty \)  
  passive states
Elapsed time
Elapsed time

\[ \text{ta}(s_i) \]

\[ e_i \]

\[ \sigma_i \]

red

yellow

green
Initialization of Initial State

\[
\begin{align*}
S(t_0) &= (s_0, 0) \\
q_{init} &= (s_0, e_0) \\
(s_0, t(a(s_0))) &= \text{red, yellow, green}
\end{align*}
\]
Elapsed time

Elapse time

\[ e \]

\[ t(a) \]

\[ e_0 \]

\[ 0 \]

\[ t \]

\[ t(a(s_0)) \]

\[ t(a(red)) \]

\[ t(a(yellow)) \]

\[ t(a(green)) \]
$S$ : set of sequential states
$S = \{\text{red, yellow, green}\}$

$\delta_{\text{int}} : S \rightarrow S$
$\delta_{\text{int}} = \{\text{red} \rightarrow \text{green}, \text{green} \rightarrow \text{yellow}, \text{yellow} \rightarrow \text{red}\}$

$ta : S \rightarrow \mathbb{R}_{0,+\infty}$
$ta = \{\text{red} \rightarrow \text{delay}_{\text{red}}, \text{green} \rightarrow \text{delay}_{\text{green}}, \text{yellow} \rightarrow \text{delay}_{\text{yellow}}\}$

$q_{\text{init}} : Q - \text{set of total states}$
$Q = \{(s,e)|s \in S, 0 \leq e \leq ta(s)\}$
$q_{\text{init}} = (\text{green}, 0)$
Abstract Syntax

\[ S = \{\text{red, yellow, green}\} \]
\[ \delta_{\text{int}} = \{ \text{red} \rightarrow \text{green}, \]
\[ \text{green} \rightarrow \text{yellow}, \]
\[ \text{yellow} \rightarrow \text{red}\} \]
\[ ta = \{\text{red} \rightarrow \text{delay}_{\text{red}}, \]
\[ \text{green} \rightarrow \text{delay}_{\text{green}}, \]
\[ \text{yellow} \rightarrow \text{delay}_{\text{yellow}}\} \]
\[ q_{\text{init}} = (\text{green}, 0) \]

Operational Semantics

time = 0

\[
\text{current}\_\text{state} = \text{initial}\_\text{state} \\
\text{last}\_\text{time} = -\text{initial}\_\text{elapsed} \\
\text{while} \text{not} \text{termination}\_\text{condition}(): \\
\quad \text{time} = \text{last}\_\text{time} + \text{ta(\text{current}\_\text{state})} \\
\quad \text{current}\_\text{state} = \delta_{\text{int}}(\text{current}\_\text{state}) \\
\quad \text{last}\_\text{time} = \text{time}
\]

Concrete Syntax

```python
from pypdevs.DEVS import *

class TrafficLightAutonomous(AtomicDEVS):
    def __init__(self, delay_green, delay_yellow, delay_red):
        AtomicDEVS.__init__(self, "Light")
        self.state = "green"
        self.elapsed = 0
        self.delay_green = delay_green
        self.delay_yellow = delay_yellow
        self.delay_red = delay_red

    def intTransition(self):
        state = self.state
        return {"red": "green", "yellow": "red", "green": "yellow"}[state]

    def timeAdvance(self):
        state = self.state
        return {"red": self.delay_red, "yellow": self.delay_yellow, "green": self.delay_green}[state]
```

**Concrete Syntax**

from pypdevs.DEVS import *

class TrafficLightAutonomous(AtomicDEVS):
    def __init__(self, delay_green, delay_yellow, delay_red):
        AtomicDEVS.__init__(self, "Light")
        self.state = "green"
        self.elapsed = 0
        self.delay_green = delay_green
        self.delay_yellow = delay_yellow
        self.delay_red = delay_red

    def intTransition(self):
        state = self.state
        return {"red": "green", "yellow": "red", "green": "yellow"}[state]

    def timeAdvance(self):
        state = self.state
        return {"red": self.delay_red, "yellow": self.delay_yellow, "green": self.delay_green}[state]
Autonomous (with output)

\[ M = \langle Y, S, q_{init}, \delta_{int}, \lambda, t a \rangle \]

\[ S = \{\text{red, yellow, green}\} \]
\[ \delta_{int} = \{ \text{red} \rightarrow \text{green}, \]
\[ \text{green} \rightarrow \text{yellow}, \]
\[ \text{yellow} \rightarrow \text{red} \} \]
\[ q_{init} = (\text{green, 0}) \]
\[ t a = \{ \text{red} \rightarrow delay_{red}, \]
\[ \text{green} \rightarrow delay_{green}, \]
\[ \text{yellow} \rightarrow delay_{yellow} \} \]

\[ Y : \text{set of output events} \]
\[ Y = \{\text{“show_red”}, \text{“show_green”}, \text{“show_yellow”}\} \]

\[ \lambda : S \rightarrow Y^b \]
\[ \lambda = \{ \text{green} \rightarrow \{\text{“show_yellow”}\}, \]
\[ \text{yellow} \rightarrow \{\text{“show_red”}\}, \]
\[ \text{red} \rightarrow \{\text{“show_green”}\} \} \]
Abstract Syntax

\[ S = \{ \text{red, yellow, green} \} \]
\[ q_{\text{init}} = (\text{green, 0}) \]
\[ \delta_{\text{int}} = \{ \begin{array}{ll}
\text{red} \rightarrow \text{green}, \\
\text{green} \rightarrow \text{yellow}, \\
\text{yellow} \rightarrow \text{red}
\end{array} \} \]
\[ ta = \{ \begin{array}{ll}
\text{red} \rightarrow \text{delay}_{\text{red}}, \\
\text{green} \rightarrow \text{delay}_{\text{green}}, \\
\text{yellow} \rightarrow \text{delay}_{\text{yellow}}
\end{array} \} \]
\[ Y = \{ \begin{array}{ll}
\text{“show_red”}, \\
\text{“show_green”}, \\
\text{“show_yellow”}
\end{array} \} \]
\[ \lambda = \{ \begin{array}{ll}
\text{green} \rightarrow \begin{array}{l}
\text{[“show_yellow”]}
\end{array}, \\
\text{yellow} \rightarrow \begin{array}{l}
\text{[“show_red”]}
\end{array}, \\
\text{red} \rightarrow \begin{array}{l}
\text{[“show_green”]}
\end{array}
\end{array} \} \]

Operational Semantics

\[
\begin{align*}
\text{time} &= 0 \\
\text{current}_\text{state} &= \text{initial}_\text{state} \\
\text{last}_\text{time} &= -\text{initial}_\text{elapsed} \\
\text{while not termination}_\text{condition}(): & \\
& \text{time} = \text{last}_\text{time} + \text{ta}(\text{current}_\text{state}) \\
& \text{output}(\lambda(\text{current}_\text{state})) \\
& \text{current}_\text{state} = \delta_{\text{int}}(\text{current}_\text{state}) \\
& \text{last}_\text{time} = \text{time}
\end{align*}
\]

Concrete Syntax

```python
from pypdevs.DEVS import *

class TrafficLightWithOutput(AtomicDEVS):
    def __init__(self):
        AtomicDEVS.__init__(self, "light")
        self.observe = self.addOutPort("observer")
        ...

    def outputFnc(self):
        state = self.state
        if state == "red":
            return {self.observe: ["show_green"]}
        elif state == "yellow":
            return {self.observe: ["show_red"]}
        elif state == "green":
            return {self.observe: ["show_yellow"]}
```
```
$M = \langle X, Y, S, q_{\text{init}}, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_{a} \rangle$

$Y = \{ \text{“show\_red”}, \text{“show\_green”}, \text{“show\_yellow”} \}$

$S = \{ \text{red}, \text{yellow}, \text{green}, \text{manual} \}$

$q_{\text{init}} = (\text{green}, 0)$

$\delta_{\text{int}} = \{ \text{red} \rightarrow \text{green}, \text{green} \rightarrow \text{yellow}, \text{yellow} \rightarrow \text{red} \}$

$\lambda = \{ \text{green} \rightarrow \text{[“show\_yellow”]}, \text{yellow} \rightarrow \text{[“show\_red”]}, \text{red} \rightarrow \text{[“show\_green”]} \}$

$t_{a} = \{ \text{red} \rightarrow \text{delay\_red}, \text{green} \rightarrow \text{delay\_green}, \text{yellow} \rightarrow \text{delay\_yellow}, \text{manual} \rightarrow \infty \}$

$X: \text{set of input events}$

$X = \{ \text{“toAuto”}, \text{“toManual”} \}$

$\delta_{\text{ext}}: Q \times X^{b} \rightarrow S$

$Q = \{ (s, e) | s \in S, 0 \leq e \leq t_{a}(s) \}$

$\delta_{\text{ext}} = \{ ( \ast, \ast), \text{[“toManual”]} \rightarrow \text{“manual”}, (\text{“manual”}, \ast), \text{[“toAuto”]} \rightarrow \text{“red”} \}$
Abstract Syntax

\[ Y = \{ \text{“show\_red”}, \text{“show\_green”}, \text{“show\_yellow”} \} \]
\[ S = \{ \text{red, yellow, green, manual} \} \]
\[ q_{init} = (\text{green, 0}) \]
\[ \delta_{int} = \{ \text{red} \rightarrow \text{green}, \]
\[ \text{green} \rightarrow \text{yellow}, \]
\[ \text{yellow} \rightarrow \text{red} \} \]
\[ \lambda = \{ \text{green} \rightarrow [\text{“show\_yellow”}], \]
\[ \text{yellow} \rightarrow [\text{“show\_red”}], \]
\[ \text{red} \rightarrow [\text{“show\_green”}] \} \]
\[ ta = \{ \text{red} \rightarrow \text{delay\_red}, \]
\[ \text{green} \rightarrow \text{delay\_green}, \]
\[ \text{yellow} \rightarrow \text{delay\_yellow}, \]
\[ \text{manual} \rightarrow \infty \} \]
\[ X = \{ \text{“toAuto”}, \text{“toManual”} \} \]
\[ \delta_{ext} = \{ (\ast, \ast), [\text{“toManual”}] \rightarrow \text{“manual”}, \]
\[ (\text{“manual”}, \ast), [\text{“toAuto”}] \rightarrow \text{“red”} \]

Operational Semantics

\[ \text{time} = 0 \]
\[ \text{cur\_state} = \text{initial\_state} \]
\[ \text{last\_time} = \text{-initial\_elapsed} \]
\[ \text{while not \text{termination\_condition}():} \]
\[ \text{next\_time} = \text{last\_time} + \text{ta}(\text{cur\_state}) \]
\[ \text{if time\_next\_ev} <= \text{next\_time:} \]
\[ \text{e} = \text{time\_next\_ev} - \text{last\_time} \]
\[ \text{time} = \text{time\_next\_ev} \]
\[ \text{cur\_state} = \delta_{ext}((\text{cur\_state, e}), \text{next\_ev}) \]
\[ \text{else:} \]
\[ \text{time} = \text{next\_time} \]
\[ \text{output}(\lambda(\text{current\_state})) \]
\[ \text{current\_state} = \delta_{int}(\text{current\_state}) \]
\[ \text{last\_time} = \text{time} \]
from pypdevs.DEVS import *

class TrafficLight(AtomicDEVS):
    def __init__(self):
        AtomicDEVS.__init__(self, "light")
        self.interrupt = self.addInPort("interrupt")
        ...
        def extTransition(self, inputs):
            inp = inputs[self.interrupt][0]
            if inp == "toManual":
                return "manual"
            elif inp == "toAuto":
                if self.state == "manual":
                    return "red"
            ...

Concrete Syntax

Abstract Syntax

\[ Y = \{\text{"show_red"}, \text{"show_green"}, \text{"show_yellow"}\} \]
\[ S = \{\text{red, yellow, green, manual}\} \]
\[ q_{\text{init}} = (\text{green, 0}) \]
\[ \delta_{\text{int}} = \{\text{red} \rightarrow \text{green}, \text{green} \rightarrow \text{yellow}, \text{yellow} \rightarrow \text{red}\} \]
\[ \lambda = \{\text{green} \rightarrow [\text{"show_yellow"}], \text{yellow} \rightarrow [\text{"show_red"}], \text{red} \rightarrow [\text{"show_green"}]\} \]
\[ t_{\text{a}} = \{\text{red} \rightarrow \text{delay}_{\text{red}}, \text{green} \rightarrow \text{delay}_{\text{green}}, \text{yellow} \rightarrow \text{delay}_{\text{yellow}}, \text{manual} \rightarrow \infty\} \]
\[ X = \{\text{"toAuto"}, \text{"toManual"}\} \]
\[ \delta_{\text{ext}} = \{(\text{*, *}, [\text{"toManual"}]) \rightarrow \text{"manual"}, (\text{("manual", *}, [\text{"toAuto"}]) \rightarrow \text{"red"}\} \]
\[ Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\} \]

\[ e = 0 \quad 0 < e < ta(s) \quad e = ta(s) \]

\[ \delta_{\text{conf}} : S \times X^b \rightarrow S \]
from pypdevs.DEVS import *

class TrafficLight(AtomicDEVS):
    ...

    def confTransition(self, inputs):
        self.elapsed = 0.0
        self.state = self.intTransition()
        self.state = self.extTransition(inputs)
        return self.state
Full Atomic DEVS Specification

\[ M = \langle X, Y, S, q_{init}, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta \rangle \]

- \( X \) : set of input events
- \( Y \) : set of output events
- \( S \) : set of sequential states

- \( q_{init} : Q \)
  \[ Q = \{ (s, e) | s \in S, 0 \leq e \leq ta(s) \} \]

- \( \delta_{int} : S \rightarrow S \)
- \( \delta_{ext} : Q \times X^b \rightarrow S \)
- \( \delta_{conf} : S \times X^b \rightarrow S \)

- \( \lambda : S \rightarrow Y^b \)
- \( ta : S \rightarrow \mathbb{R}_{0,+\infty} \)
\[ (\delta_{\text{ext}}((s_i, e), x), 0) \]

\[ (s_i, 0) \]

\[ e \]

\[ \delta_{\text{ext}} \]

\[ (\delta_{\text{int}}(s_i), 0) \]

\[ \text{output } \lambda(s_i) \]

\[ t_{i+e} \]

\[ t_{i+\tau(a(s_i))} \]

\[ \tau(a(s_i)) \]

\[ \delta_{\text{int}} \]
Coupled Models
\( e = 0s \)

- work 3600s
- idle 300s

\(!go\_to\_work \quad !take\_break\)
\[ C = \langle D, \{M_i\}\rangle \]
\[ \{M_i | i \in D\} \]
\[ M_i = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_{\alpha}\rangle, \forall i \in D \]
\[ C = \langle X_{self}, Y_{self}, D, \{M_i\} \rangle \]
\[ \{M_i| i \in D \} \]
\[ M_i = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle, \forall i \in D \]
\[ C = \{X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}\} \]
\[ \{M_i|i \in D\} \]
\[ M_i = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t\alpha\rangle, \forall i \in D \]
\[ \{I_i|i \in D \cup \{self\}\} \]
\[ \forall i \in D \cup \{self\} : I_i \subseteq D \cup \{self\} \]
\[ \forall i \in D \cup \{self\} : i \notin I_i \]

\[ e = 0s \]

\[ \text{go_to_work}! \]

\[ \text{take_break}! \]

\[ e = 0\infty \]

\[ \text{show_red}! \]

\[ \text{delay_red}! \]

\[ \text{toAuto}! \]

\[ \text{toManual}! \]

\[ \text{toManual}! \]

\[ \text{show_green}! \]

\[ \text{delay_green}! \]

\[ \text{toManual}! \]

\[ \text{delay_yellow}! \]

\[ \text{show_yellow}! \]

\[ \text{delay_yellow}! \]

\[ \text{delay_green}! \]

\[ \text{show_green}! \]

\[ \text{delay_green}! \]

\[ \text{toManual}! \]
\[ C = \left\langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \right\rangle \]
\[ \{M_i\mid i \in D\} \]
\[ M_i = \left\langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \right\rangle, \forall i \in D \]
\[ \{I_i\mid i \in D \cup \{self\}\} \]
\[ \forall i \in D \cup \{self\} : I_i \subseteq D \cup \{self\} \]
\[ \forall i \in D \cup \{self\} : i \notin I_i \]
\[ \{Z_{i,j}\mid i \in D \cup \{self\}, j \in I_i\} \]
\[ Z_{self,j} : X_{self} \rightarrow X_j, \forall j \in D \]
\[ Z_{i,\text{self}} : Y_i \rightarrow Y_{\text{self}}, \forall i \in D \]
\[ Z_{i,j} : Y_i \rightarrow X_j, \forall i, j \in D \]

take_break \rightarrow \text{toAuto}

go_to_work \rightarrow \text{toManual}
from pypdevs.DEVS import *
from trafficlight import TrafficLight
from policeman import Policeman

def translate(in_evt):
    mapping = {"take_break": "toAuto",
                "go_to_work": "toManual"}
    return mapping[in_evt]

class TrafficLightSystem(CoupledDEVS):
    def __init__(self):
        CoupledDEVS.__init__(self, "system")
        self.light = self.addSubModel(TrafficLight())
        self.police = self.addSubModel(Policeman())
        self.connectPorts(self.police.out, self.light.interrupt, translate)
Closure under Coupling
\[ CM = \langle X_{self}, Y_{self}, D, \{M_i\}, \{J_i\}, \{Z_{i,j}\} \rangle \]
\[ M_i = \langle X_i, Y_i, S_i, \delta_{int,i}, \delta_{ext,i}, \lambda_i, t_{ai} \rangle \forall i \in D \]
Hierarchical Simulator
## DEVS Semantics

<table>
<thead>
<tr>
<th>Atomic DEVS</th>
<th>Coupled DEVS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operational Semantics</strong></td>
<td><strong>Denotational Semantics</strong></td>
</tr>
<tr>
<td>Abstract Simulator</td>
<td>[1]</td>
</tr>
<tr>
<td>Hierarchical Simulator</td>
<td>Closure under Coupling</td>
</tr>
</tbody>
</table>

Conclusions

- Atomic DEVS
- Coupled DEVS
- Closure under coupling
- Abstract Simulator
Examples

A small trafficModel and corresponding trafficExperiment file is included in the examples folder of the PyPDEVS distribution. This (completely working) example is slightly too big to use as a first introduction to PyPDEVS and therefore this page will start with a very simple example.

For this, we will first introduce a simplified queue model, which will be used as the basis of all our examples. The complete model can be downloaded: queue_example_classic.py.

This section should provide you with all necessary information to get you started with creating your very own PyPDEVS simulation. More advanced features are presented in the next section.

Generator

Somewhat simpler than a queue even, is a generator. It will simply create a message to send after a certain delay and then it will stop doing anything.

Informally, this would result in a DEVS specification as:

- Time advance function returns the waiting time to generate the message, infinity after the message was created
- Output function returns the newly generated message
- External transition function will never happen (as there are no inputs)

http://msdl.cs.mcgill.ca/projects/PythonPDEVS