



NAVAL
POSTGRADUATE
SCHOOL

Introduction to PyML

2022 INCOSE San Diego Mini-Conference

December 3, 2022

Dr. Raymond Madachy
Naval Postgraduate School
Dept. of Systems Engineering

rjmadach@nps.edu



Acknowledgments

- This research is supported by:
 - Naval Postgraduate School Foundation for *Open Source System Modeling Library* research seed project
 - Marine Corps Air Station (MCAS) Miramar for *Optimize the Military and Civilian Workforce with Respect to both Cost and Readiness* research project
- The following NPS faculty and students have provided critical feature suggestions and testing support:
 - Dr. John (Mike) Green
 - Nora Aftel
 - Patricia Gomez



Python Modeling Library (PyML) Goals

- Lower access barrier to system modeling with open-source tool environment
 - Harness power of extensive Python scientific computing and utility libraries (bootstrap don't reinvent)
- Provide integrated capabilities for systems modeling, analysis, documentation and code generation
 - SysML, other SE model types and analysis methods
- Be digital engineering compliant with single source of truth across model set
- Compatibility of everything on desktop OS's and in cloud



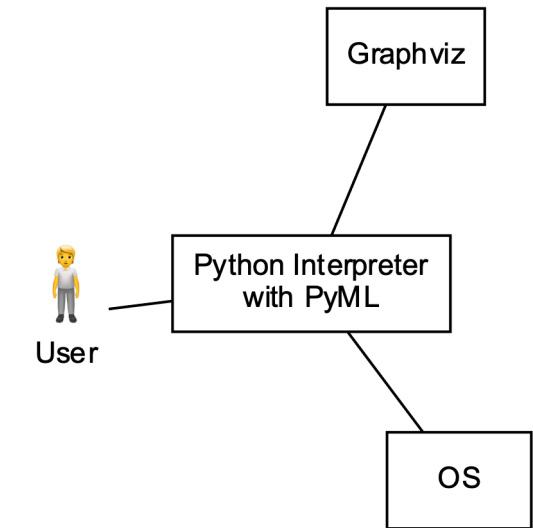
General Usage Features

- Use Python for modeling to intersperse model data, system analysis and documentation
- Model data can be inline code, read from or to external files
- Round-trip engineering to support rapid *iterative* development
 - Change a model element and all others affected are automatically updated to maintain consistency across model set
 - Automatic document re-generation with all changes
- Inherent configuration management automation
 - All artifacts for a project can be developed and controlled with standard version control system such as GitHub for small to large teams.
 - All models, diagrams, and simulations are specified in text files supporting standard tools for version differencing and reconciliation.

```
import pyml

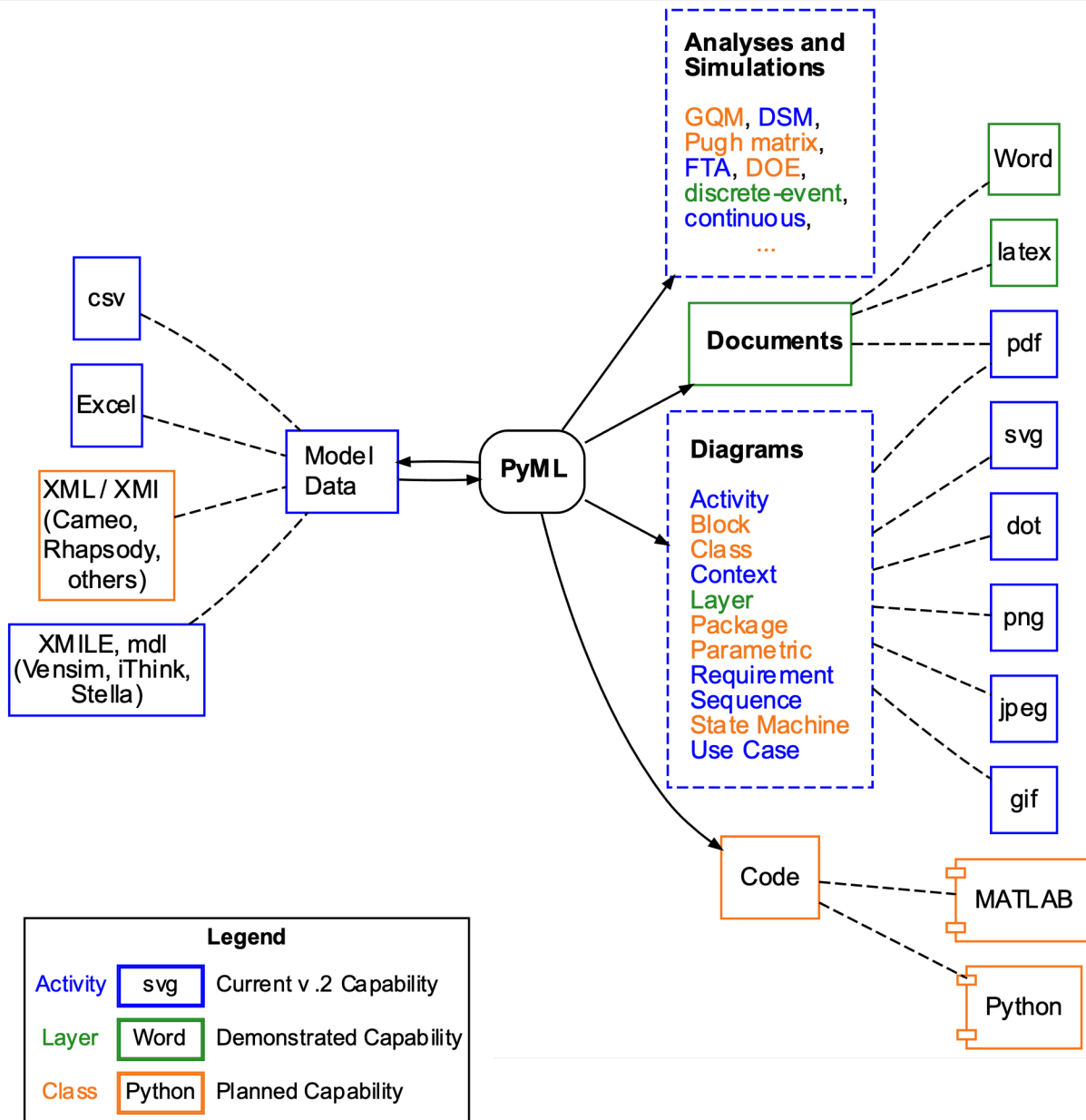
# system model
system_name = "Python Interpreter with PyML"
external_actors = ["User", "OS", "Graphviz"]

# create context diagram
pyml.context_diagram(system_name, external_actors)
```





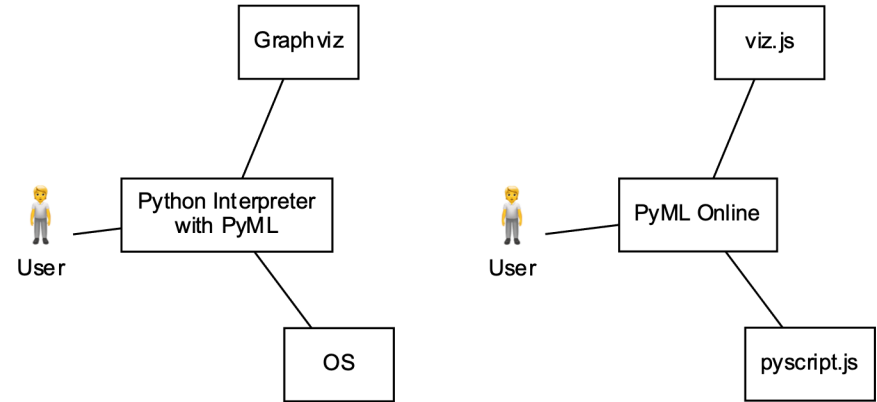
Inputs and Outputs





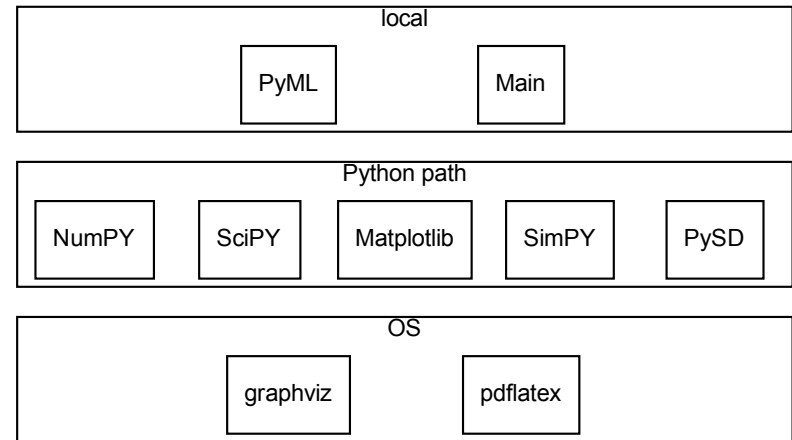
PyML Context and Architecture

- Desktop usage requires Python version 3.8 or higher.
- Built on top of open-source:
 - *Graphviz* is required to generate diagrams.
 - The *Matplotlib* package is required for graphical plots.
 - *NumPy* numerical computing package is used for model analysis and some plotting features.
 - *SciPy* is a scientific computing library
 - *SimPy* is a discrete event simulation framework
 - *PySD* with *Pandas* for the system dynamics simulation engine
 - *pdflatex* optionally required to compile latex files and generate pdfs



Offline

Online



Desktop Layers



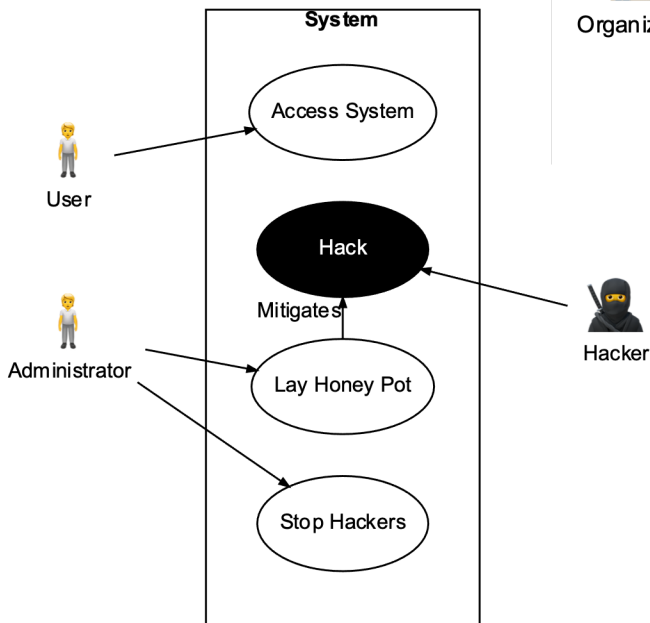
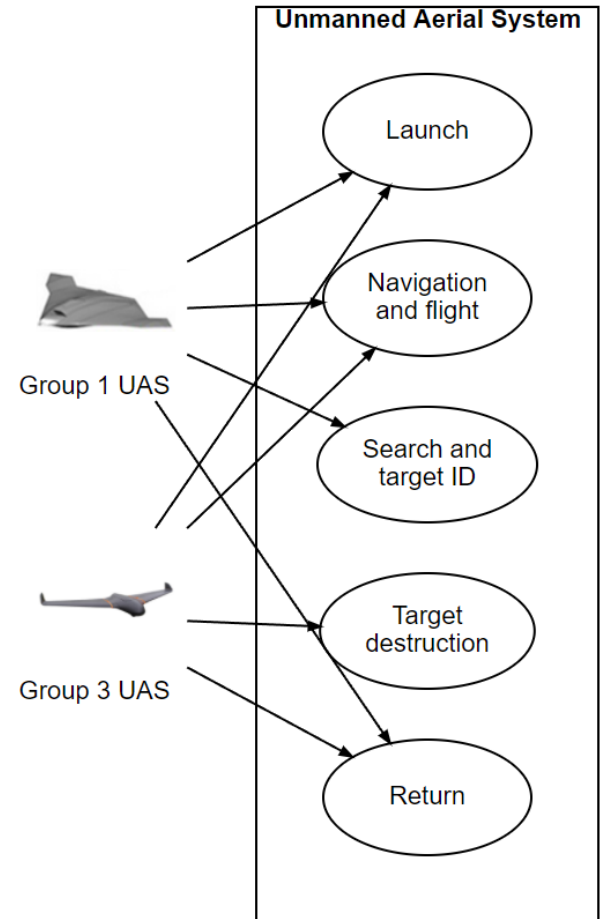
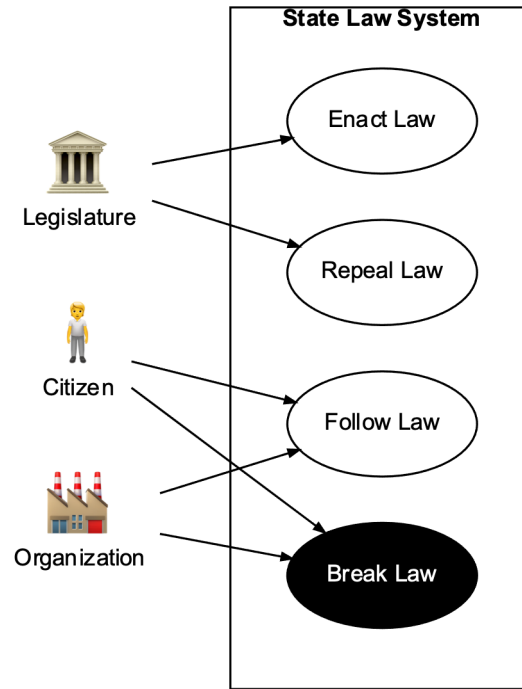
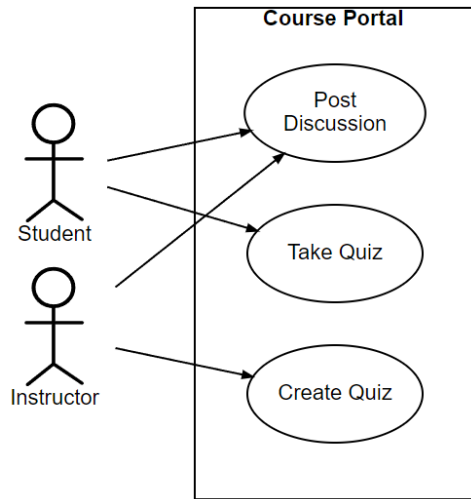
Feature Plans

- V 1.0 release plan is to cover all SysML

Feature	v. .2 Current	v. 1.0 Planned	v. 1.0+ Planned
<u>SysML</u>			✓
Activity diagram	✓ (partial)	✓	✓ simulatable
Package diagram		✓	✓
Use case diagram	✓	✓	✓
Requirements diagram	✓ (via Tree)	✓	✓
Sequence diagram	✓	✓	✓ simulatable
Block diagram		✓	✓
State machine diagram		✓	✓ simulatable
Parametric diagram		✓	✓
<u>Other</u>			
Context diagram	✓	✓	✓
Qualitative fault tree diagram	✓	✓	✓
Quantitative fault tree diagram	✓	✓	✓ simulatable
Fault tree cutsets	✓	✓	✓
Class diagram			✓
Layer diagram			✓
Orthogonal Variability Model (OVM) diagram			✓
Critical path analysis and diagram	✓	✓	✓
Work Breakdown Structure (WBS) diagram	✓	✓	✓
Design Structure Matrix (DSM) diagram	✓	✓	✓
N2 diagram			✓
System dynamics modeling and simulation	✓	✓	✓ diagrams
TBD		✓	✓

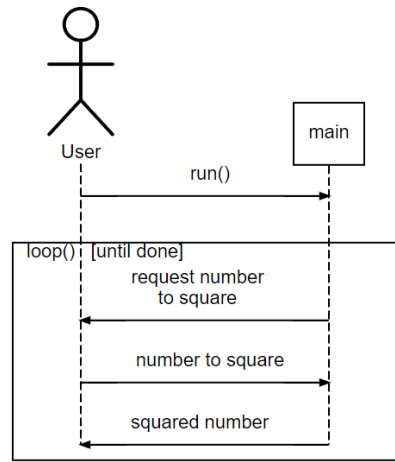
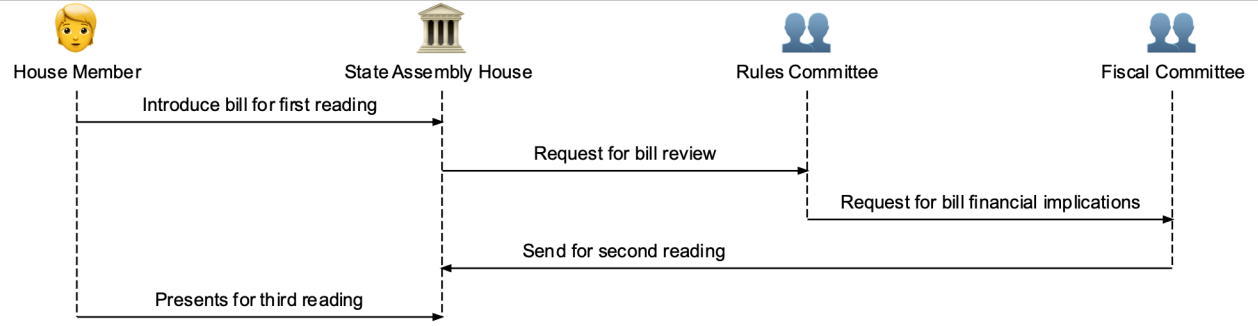
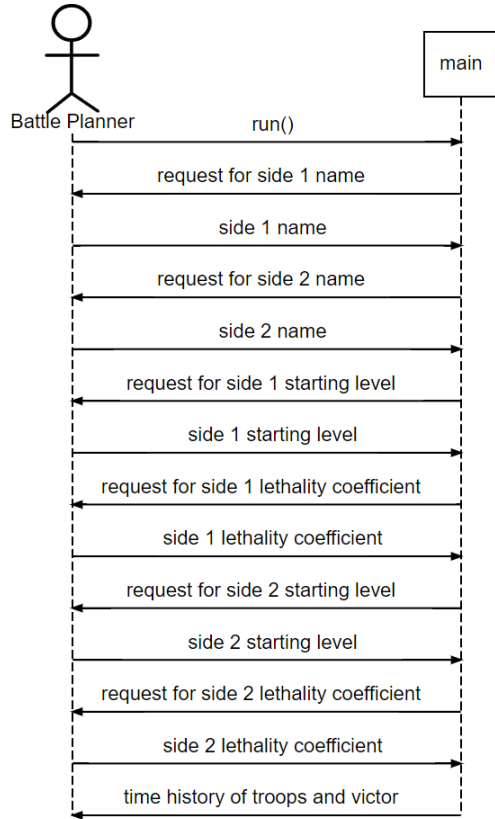


Use Case Diagram Examples





Sequence Modeling Examples





Battle Simulator Inline Model Example

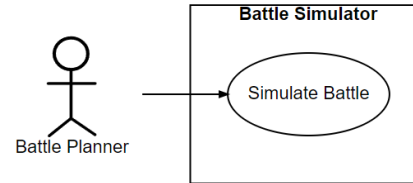
Input

```
import pyml

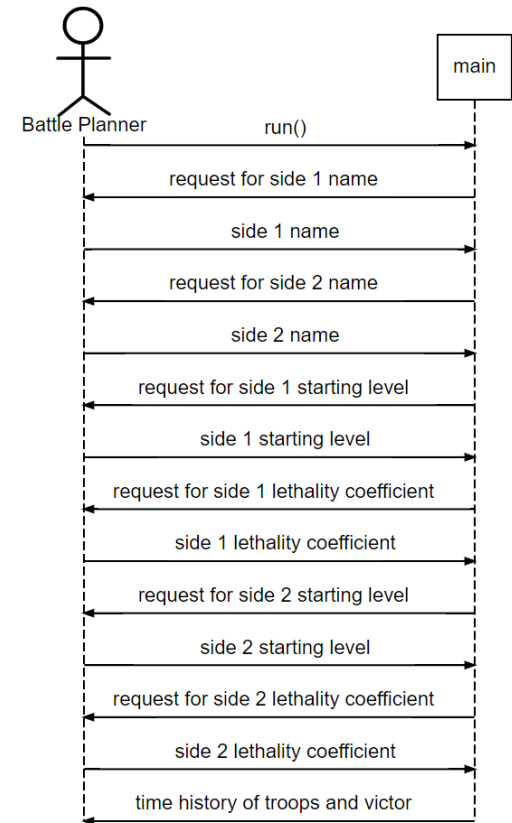
# system model
system_name = "Battle Simulator"
actors = ['Battle Planner']
use_cases = ['Simulate Battle']
interactions = [('Battle Planner', 'Simulate Battle')]
use_case_relationships = []

actions = [
('Battle Planner', 'main', 'run()'),
('main', 'Battle Planner', 'request for side 1 name'),
('Battle Planner', 'main', 'side 1 name'),
('main', 'Battle Planner', 'request for side 2 name'),
('Battle Planner', 'main', 'side 2 name'),
('main', 'Battle Planner', 'request for side 1 starting level'),
('Battle Planner', 'main', 'side 1 starting level'),
('main', 'Battle Planner', 'request for side 1 lethality coefficient'),
('Battle Planner', 'main', 'side 1 lethality coefficient'),
('main', 'Battle Planner', 'request for side 2 starting level'),
('Battle Planner', 'main', 'side 2 starting level'),
('main', 'Battle Planner', 'request for side 2 lethality coefficient'),
('Battle Planner', 'main', 'side 2 lethality coefficient'),
('main', 'Battle Planner', 'time history of troops and victor'),
]

# create diagrams
pyml.use_case_diagram(system_name, actors, use_cases, interactions,
use_case_relationships, filename=system_name+'use_case_diagram.pdf')
pyml.sequence_diagram(system_name, actions, filename=system_name
+'sequence_diagram.pdf')
```



Output

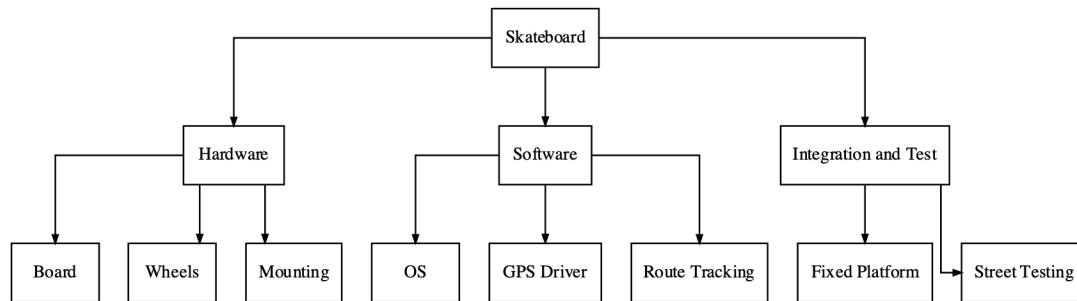




Project Management Modeling (1/2)

```
# project work breakdown structure
wbs_decompositions = [('Skateboard', 'Hardware'), ('Skateboard',
'Software'), ('Skateboard', 'Integration and Test'), ('Hardware',
'Board'), ('Hardware', 'Wheels'), ('Hardware', 'Mounting'),
('Software', 'OS'), ('Software', 'GPS Driver'), ('Software', 'Route
Tracking'), ('Integration and Test', 'Fixed Platform'), ('Integration
and Test', 'Street Testing')]

# create diagram
pyml.wbs_diagram(wbs_decompositions)
```

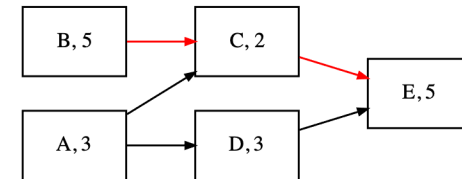


```
# project tasks
tasks = [('A', {'Duration': 3}),
('B', {'Duration': 5}),
('C', {'Duration': 2}),
('D', {'Duration': 3}),
('E', {'Duration': 5})]

task_dependencies = [('A', 'C'),
('B', 'C'),
('A', 'D'),
('C', 'E'),
('D', 'E')]

# create diagram
pyml.critical_path_diagram(tasks, task_dependencies)
```

The critical path is: ['B', 'C', 'E'] for a project duration of 12 days.



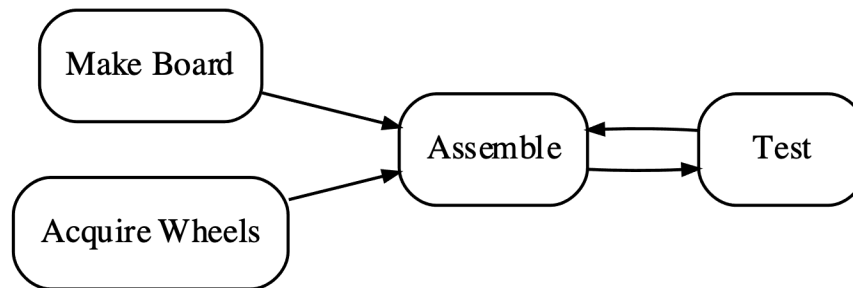


Project Management Modeling (2/2)

```
tasks = ['Make Board', 'Acquire Wheels', 'Assemble', 'Test']  
task_dependencies = [('Make Board', 'Assemble'), ('Acquire Wheels',  
'Assemble'), ('Assemble', 'Test'), ('Test', 'Assemble')]  
pyml.design_structure_matrix(tasks, task_dependencies)
```

	Make Board	Acquire Wheels	Assemble	Test
Make Board				
Acquire Wheels				
Assemble	X	X		X
Test			X	

```
pyml.activity_diagram(task_dependencies)
```

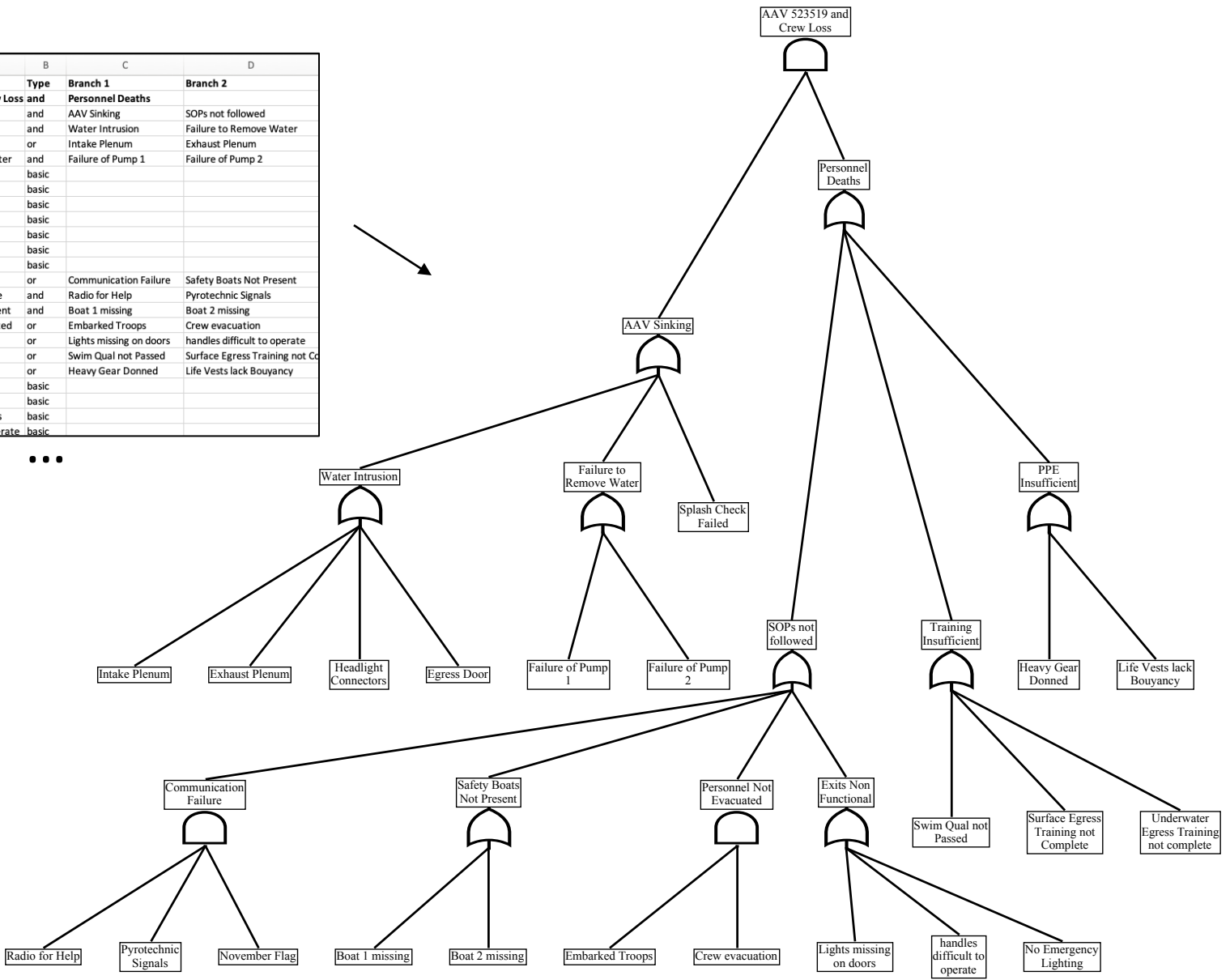




AAV Disaster Fault Tree from Excel

	A	B	C	D
1	Name	Type	Branch 1	Branch 2
2	AAV 523519 and Crew Loss	and	Personnel Deaths	
3	Personnel Deaths	and	AAV Sinking	SOPs not followed
4	AAV Sinking	and	Water Intrusion	Failure to Remove Water
5	Water Intrusion	or	Intake Plenum	Exhaust Plenum
6	Failure to Remove Water	and	Failure of Pump 1	Failure of Pump 2
7	Intake Plenum	basic		
8	Exhaust Plenum	basic		
9	Headlight Connectors	basic		
10	Failure of Pump 1	basic		
11	Failure of Pump 2	basic		
12	Egress Door	basic		
13	Splash Check Failed	basic		
14	SOPs not followed	or	Communication Failure	Safety Boats Not Present
15	Communication Failure	and	Radio for Help	Pyrotechnic Signals
16	Safety Boats Not Present	and	Boat 1 missing	Boat 2 missing
17	Personnel Not Evacuated	or	Embarked Troops	Crew evacuation
18	Exits Non Functional	or	Lights missing on doors	handles difficult to operate
19	Training Insufficient	or	Swim Qual not Passed	Surface Egress Training not C
20	PPE Insufficient	or	Heavy Gear Donned	Life Vests lack Bouyancy
21	Embarked Troops	basic		
22	Crew evacuation	basic		
23	Lights missing on doors	basic		
24	handles difficult to operate	basic		

...

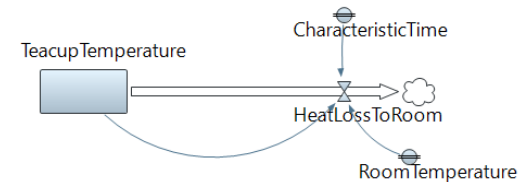
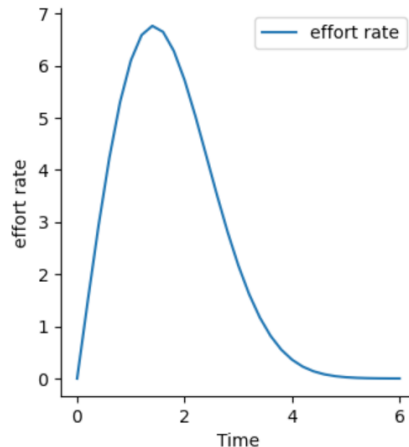
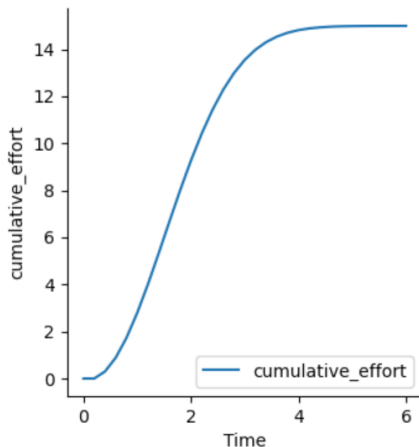




System Dynamics Modeling and Simulation

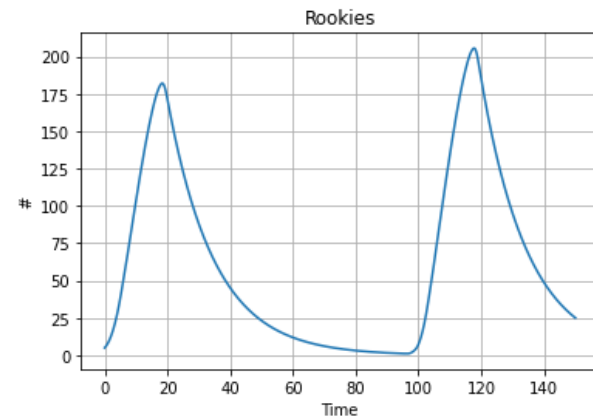
- Functions provide high level interface for model composition and execution with PySD model reader and simulation engine.
- Interoperable with Vensim, iThink/Stella and AnyLogic with xmile model format.

```
1 # Rayleigh curve staffing model
2
3 model_init(start=0, stop=6, dt=.2)
4
5 add_stock("cumulative_effort", 0, inflows=["effort rate"])
6 add_flow("effort rate", "learning_function * (estimated_total_effort - cumulativ
7 add_auxiliary("learning_function", "manpower_buildup_parameter * time")
8 add_auxiliary("manpower_buildup_parameter", .5)
9 add_auxiliary("estimated_total_effort", 15)
10
11 model_run()
12 plot_output('cumulative_effort', 'effort rate')
```



```
model = read_xmile('teacup.xmile')
model_run()
```

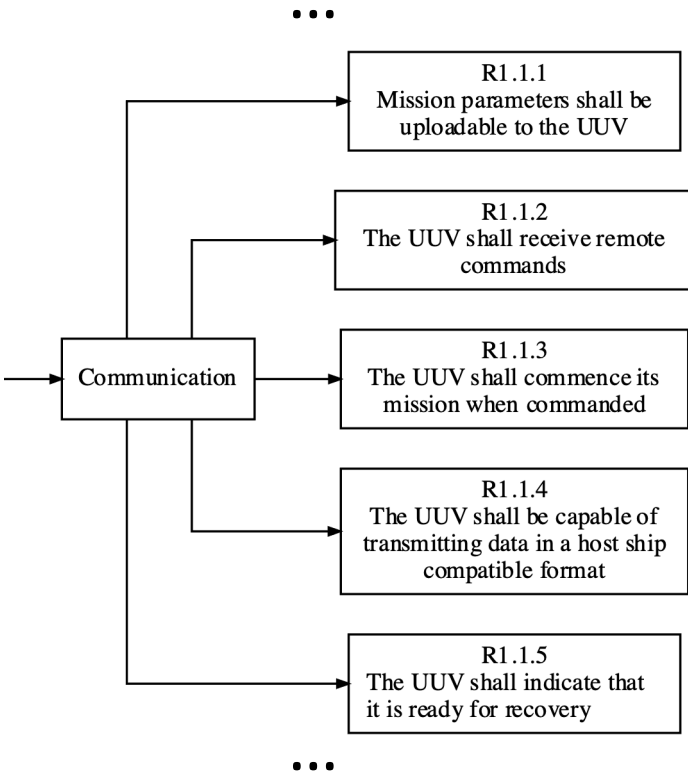
```
model = read_vensim('workforce.mdl')
model.run()
plot_output("Rookies")
```





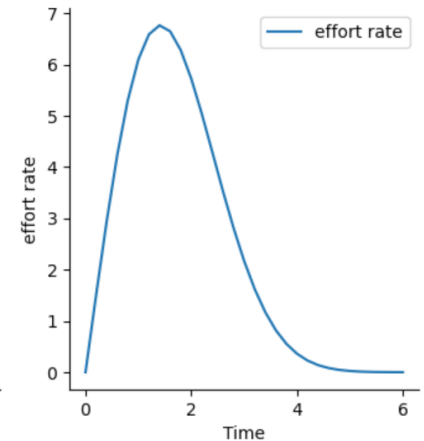
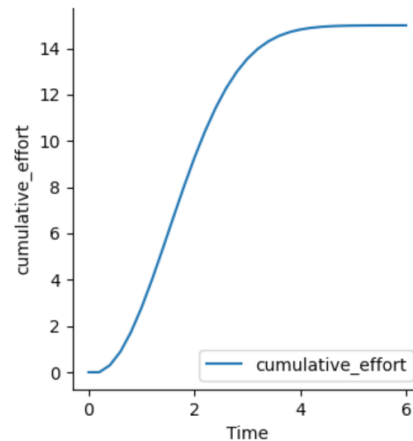
Integrated Requirements and Effort Models

```
requirements = [("ISR", "Communication"),
                ("Communication", "R1.1.1 Mission parameters shall
                uploadable to the UUV"),
                ("Communication", "R1.1.2 The UUV shall receive remote
                commands"),
                ("Communication", "R1.1.3 The UUV shall commence its
                mission when commanded"),
                ("Communication", "R1.1.4 The UUV shall be capable of
                transmitting data in a host ship compatible format"),
                ("Communication", "R1.1.5 The UUV shall indicate that
                it is ready for recovery")
                ]
```



```

1 # effort model
2 def cosysmo(size, EAF=1):
3     return .254*EAF*size**1.06, 1.5*(.254*EAF*size**1.06)**.33
4
5 # effort estimate from requirements
6 requirements_count = len(requirements)
7 effort, schedule = cosysmo(requirements_count)
8
9 # Rayleigh curve staffing model
10
11 model_init(start=0, stop=6, dt=.2)
12
13 add_stock("cumulative_effort", 0, inflows=["effort rate"])
14 add_flow("effort rate", "learning_function * (estimated_total_effort - cumulativ
15 add_auxiliary("learning_function", "manpower_buildup_parameter * time")
16 add_auxiliary("manpower_buildup_parameter", .5)
17 add_auxiliary("estimated_total_effort", effort)
18
19 model_run()
20 plot_output('cumulative_effort', 'effort rate')
  
```





Customization and Shortcuts

- Styling

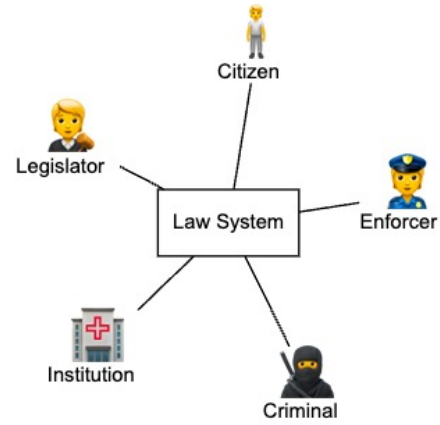
- Custom icons with unicode
- APIs support color options, formatting, word wrap size, and more.

- Shortcuts

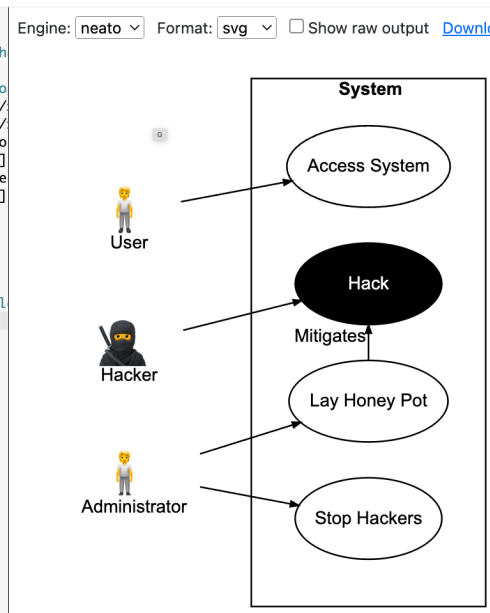
- Use tuples for node fan-in and fan-out connections to alleviate redundancy

- *Graphviz online* tool enables customization of dot markup at http://pym1.fun/graphviz_online

```
system = 'Law System'  
actors_and_external_systems = [('Institution', '🏥'), ('Criminal', '👤'),  
('Citizen', '👤'), ('Enforcer', '👮'), ('Legislator', '👤')]  
pym1.context_diagram(system, actors_and_external_systems)
```



```
1 # cyber security misuse case honey pot  
2 digraph G {  
3   node [color=black fontname=arial fontsize=11 width  
4   edge [arrowsize=.5 fontname=arial fontsize=11]  
5   "System" [label=<<b>System</b>> height=4.5 labello  
6   "User" [label=<<font point-size="30">👤</font><br/>  
7   Hacker [label=<<font point-size="30">👤</font><br/>  
8   Administrator [label=<<font point-size="30">👤</fo  
9   "Access System" [height=.7 pos="0, 41" width=1.25]  
10  Hack [height=.7 pos="0, 31" width=1.25 style=fill  
11  "Lay Honey Pot" [height=.7 pos="0, 21" width=1.25]  
12  "Stop Hackers" [height=.7 pos="0, 11" width=1.25]  
13  "User" -> "Access System"  
14  Administrator -> "Lay Honey Pot"  
15  Administrator -> "Stop Hackers"  
16  Hacker -> Hack  
17  "Lay Honey Pot" -> Hack [label="Mitigates" label  
18 }
```





Example Document Generation Including Diagrams

Input

```
from pyml import *

# system model
system_name = "Battle Simulator"
actors = ['Battle Planner']
use_cases = ['Simulate Battle']
interactions = [('Battle Planner', 'Simulate Battle')]
use_case_relationships = []

actions = [
    ('Battle Planner', 'main', 'run()'),
    ('main', 'Battle Planner', 'request for side 1 name'),
    ('Battle Planner', 'main', 'side 1 name'),
    ('main', 'Battle Planner', 'request for side 2 name'),
    ('Battle Planner', 'main', 'side 2 name'),
    ('main', 'Battle Planner', 'request for x starting level'),
    ('Battle Planner', 'main', 'x starting level'),
    ('main', 'Battle Planner', 'request for x lethality coefficient'),
    ('Battle Planner', 'main', 'x lethality coefficient'),
    ('main', 'Battle Planner', 'request for y starting level'),
    ('Battle Planner', 'main', 'y starting level'),
    ('main', 'Battle Planner', 'request for y lethality coefficient'),
    ('Battle Planner', 'main', 'y lethality coefficient'),
    ('main', 'Battle Planner', 'time history of troops and victor'),]

# create diagrams
use_case_diagram(system_name, actors, use_cases, interactions,
use_case_relationships, filename=system_name+'_use_case_diagram.pdf')
sequence_diagram(system_name, actors, actions, filename=system_name
+'_sequence_diagram.pdf')

# generate document
latex_create(system_name + "Model Description")
section("Introduction")
latex_string(f"The {system_name} system is used by the {list_elements(actors)}
actor. Its use case diagram is in Figure \\ref{{Use Case Diagram}} and
sequence diagram in Figure \\ref{{Sequence Diagram}}.")
figure("Use Case Diagram", system_name+'_use_case_diagram.pdf')
figure("Sequence Diagram", system_name+'_sequence_diagram.pdf')
latex_write(system_name + "Model Description.pdf")
```

Output

Battle Simulator Model Description

1 Introduction

The Battle Simulator system is used by the Battle Planner actor. Its use case diagram is in Figure 1 and sequence diagram in Figure 2.

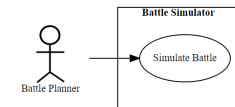


Figure 1: Use Case Diagram

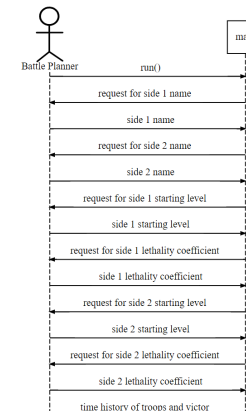


Figure 2: Sequence Diagram



Initial Online Capabilities

- Test online before downloading
- Use online as-is and manage model files locally

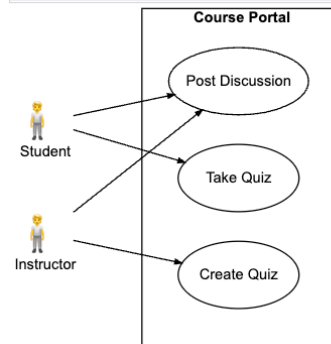
<http://pymf.fun/online/sysml.html>

SysML Diagram Scratchpad

Enter Python statements in code cells and click the green run button or hit shift-enter to create diagrams. Additional diagrams can be created in the blank code cells at the end that are automatically generated after runs.

Use Case Diagram

```
1 # system model
2 system_name = "Course Portal"
3 actors = ['Student', 'Instructor']
4 use_cases = ['Post Discussion', 'Take Quiz', 'Create Quiz']
5 interactions = [('Student', 'Post Discussion'), ('Instructor', 'Post Discussion'), ('
6 ('Instructor', 'Create Quiz')]
7 use_case_relationships = []
8
9 # create diagram
10 use_case_diagram(system_name, actors, use_cases, interactions, use_case_relationships)
```

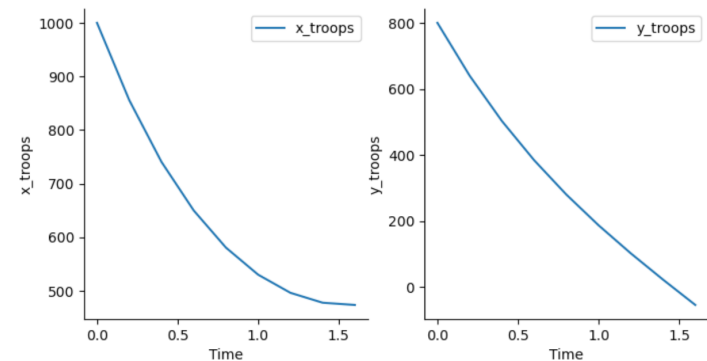


http://pymf.fun/online/system_dynamics.html

System Dynamics Demonstrations

These introductory models can be edited and run with the green play buttons. The PyML function names are tentative. See [GitHub](https://github.com) for the latest PyML library and future examples at pymf.fun.

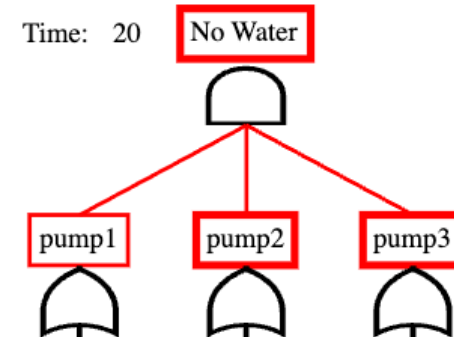
```
1 # Battle Simulator using Lanchester's Law for Aimed Fire
2
3 model_init(start=0, stop=1.5, dt=.2)
4
5 add_stock("x_troops", 1000, outflows=["x_attrition"])
6 add_flow("x_attrition", "y_troops*y_lethality")
7 add_auxiliary("x_lethality", .8)
8
9 add_stock("y_troops", 800, outflows=["y_attrition"])
10 add_flow("y_attrition", "x_troops*x_lethality")
11 add_auxiliary("y_lethality", .9)
12
13 model_run()
14 plot_output('x_troops', 'y_troops')
```





Prototyped Upcoming Features

- Activity model diagrams with more node types
- State model diagrams
- Orthogonal variability model diagrams
- [Dynamic fault tree simulation](#)





Future Work and Capabilities

- Publish library on Python Package Index (PyPI) software repository
- Additional object-oriented API
 - Foster reuse, adoption and increase sophistication
 - Can alleviate manual bookkeeping across models
- Graphical editor for diagrams in browser using open-source JavaScript
- Natural language extensions
 - E.g., write concise sentences for use case scenario and sequence model interactions using simple grammar rules and keywords
- Code generation from sequence and activity models
- See <http://pyml.fun> for more information.



References

- *Python Modeling Library (PyML)*, <http://pyml.fun>, Accessed December 2, 2022
- *Python Modeling Library (PyML) Repository*, <https://github.com/madachy/PyML>, Accessed December 2, 2022
- *sysml.org, SysML Open Source Project - What is SysML? Who created SysML?* <https://sysml.org/>, Accessed December 7, 2021
- R. Giachetti. “Digital Engineering.” in SEBoK Editorial Board. 2021. The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 2.5. Hoboken, NJ: The Trustees of the Stevens Institute of Technology, https://www.sebokwiki.org/wiki/Digital_Engineering, Accessed December 7, 2021
- U.S. Department of Defense CIO, *DoD Open Source Software FAQ*, <https://dodcio.defense.gov/open-source-software-faq/>, Accessed December 7, 2021



Backup



Available Functions

```
# Version 0.2 function calls
```

```
# SysML and related
```

```
context_diagram(system, external_systems, filename=None, format='svg', engine='neato')  
activity_diagram(element_dependencies, filename=None, format='svg')  
use_case_diagram(system_name, actors, use_cases, interactions, use_case_relationships,  
filename=None, format='svg')  
sequence_diagram(system_name, actors, objects, actions, filename=None, format='svg')
```

```
# project modeling
```

```
critical_path_diagram(tasks, task_dependencies, filename=None, format='svg')  
design_structure_matrix(elements, element_dependencies, filename=None, format='svg')  
wbs_diagram(decompositions, filename=None, format='svg', rankdir='TB')
```

```
# generic
```

```
tree(element_dependencies, filename=None, format='svg')
```

```
# fault trees
```

```
fault_tree_diagram(ft, filename=None, format='svg')  
read_fault_tree_excel(filename)  
draw_fault_tree_diagram_quantitative(ft, filename=None, format='svg'):  
fault_tree_cutsets(fault_tree)
```

```
# system dynamics
```

```
model_init()  
add_stock()  
add_flow()  
add_auxiliary()  
model_run()  
plot_output()
```



Example API Specifications

pyml.sequence_diagram

```
pyml.sequence_diagram(system_name, actors, objects, actions, filename=None, format='svg')
```

Draw a sequence diagram.

Parameters:: **system_name** : *string*

The name of the system to label the diagram.

actors : *list of strings*

Names of the outside actors that participate in the activity sequence in a list.

objects : *list of strings*

Names of the system objects that participate in the activity sequence in a list.

actions : *list of tuples*

A chronologically ordered list describing the sequence of actions to be drawn. Each action is a tuple containing the action source, target and action name (or data/control passed) in the form ("source", "target", "action name") indicating a labeled horizontal arrow drawn between them.

filename : *string, optional*

A filename for the output not including a filename extension. The extension will be specified by the format parameter.

format : *string, optional*

The file format of the graphic output. Note that bitmap formats (png, bmp, or jpeg) will not be as sharp as the default svg vector format and most particularly when magnified.

Returns:: **g** : *graph object view*

Save the graph source code to file, and open the rendered result in its default viewing application. PyML calls the [Graphviz API](#) for this.



Example Modeler Scenario for Roundtrip Digital Engineering

- Joe is developing a system architecture and wants to make changes for integrated models in one place with no extra manual steps to recompute analyses, regenerate all model artifacts for other stakeholders, and communicate important change impacts. PyML library functions are used in the modeling and documentation. All model data and analysis programs must be configuration controlled to adhere to a common baseline.
- The distributions of some system parameters need to be revised that will affect requirements, performance and cost models. Simulations assess the system availability and reliability, perform a hazard analysis, estimate cost, and the Measures of Effectiveness (MOEs) are aggregated in a weighted criteria matrix.
- He accesses the shared project repository on GitHub and updates the parameters in a main configuration file. He commits the changes back into GitHub.
- He wants the changes to trigger a common script that reconciles the models, recomputes all simulations with the updated parameter values, and produces an updated set of documents online for other stakeholders. All affected visualizations need to be regenerated and inserted into iterative project documents. The scripts need to call the requisite programs for analysis and documentation using the library functions with updated model data.
- Errors and warning notices are to be provided for any broken or inconsistent models. E.g., a performance or cost threshold is not being met, or a previously working simulation model fails to execute properly with the new parameter values.
- He wants the stakeholders who specified thresholds for affected MOEs to automatically get notice. He expects this will occur if the requirements data contains the source of each requirement.



DoD Business Case

- Anecdotal evidence indicates the Pareto Law holds for costly MBSE vendor tools: about 90% of users use only 10% of their features.
- Open source software is fully permissible and encouraged in the DoD
- DoD Open Source Software FAQ:
 - <https://dodcio.defense.gov/open-source-software-faq/>