Aiding the Task of Process-Based Modeling with ProBMoTViz

Gjorgi Peev, Nikola Simidjievski, Sašo Dzeroski
Jozef Stefan Institute & Jozef Stefan International Postgraduate School
Jamova cesta 39
Ljubljana, Slovenia
gjorgi.peev@ijs.si, nikola.simidjievski@ijs.si, saso.dzeroski@ijs.si

ABSTRACT: *Process-based modeling (PBM) is an equation discovery approach for automated modeling of dynamical systems, which takes at input substantial expert knowledge and measured data of the observed system. The resulting process-based models offer both high-level representation (in terms of building blocks / model components, i.e., entities and processes) and low-level representation (a set of ordinary differential equations). ProBMoT, a software platform for modeling, parameter estimation, and simulation of process-based models, is the latest implementation of the process-based modeling approach. While ProBMoT has been successfully applied to the task of modeling dynamical systems, compared to other modeling and simulation software, it is substantially behind in terms of user-friendliness. The goal of the present work is to overcome this limitation of ProBMoT. We design and implement an extension of ProBMoT, named ProBMoTViz, which is a platform consisting of a GUI (Graphical User Interface) for ProBMoT and a PBM Visualizer for process based models. We evaluate the versatility of ProBMoTViz on example case studies.*

Keywords: Process-based modeling, Graphic User Interface

Received: 10 September 2018, Revised 24 October 2018, Accepted 5 November 2018

1. Introduction

ProBMoT [4, 10] is the latest implementation of the process-based modeling approach [3, 6] for automated modeling of dynamic systems. Given a background knowledge, modeling constraints and measured data at input, ProBMoT constructs completely defined process-based models, represented with entities and processes.

ProBMoT has been successfully applied to a variety of modeling tasks in a number of real-world domains, such as aquatic ecosystems [4]; population dynamics [5]; biological systems [11]; oscillatory systems [8]; as well as predicting future behavior of the system at hand [9]. Unlike other modeling and simulation tools [7], ProBMoT is a domain-free tool and can be applied to any modeling task that involves model structure identification and/or parameter estimation. However, it still straggles behind these tools in terms of graphical/visual representation of the constructed models, comprehensibility of the output for a broader userbase as well as user-friendliness when it comes to preparing and running a PBM task. User feedback indicates that a GUI and a visual representation of process-based models can overcome these obstacles.

In this work, we set out to overcome the usability limitations of ProBMoT, i.e., to expand its user scope by developing and implementing an extension for it. In particular, we propose ProBMoTViz, a software platform which includes a Graphical User Interface (GUI) for ProBMoT and a PBM Visualizer for process-based models. On one hand, the GUI supports the basic operations of (automated) modeling dynamical systems in terms of providing appropriate input for the modeling and examining the outputs thereof. On the other hand, the PBM Visualizer illiterates the (currently textual) output models with a higher-level visual representation, that better communicates with the domain experts.

2. Problem Definition

ProBMoT addresses the task of automated modeling in terms of automated search of the appropriate model structure and estimating its parameter values. The input to the tool includes several input files: (1) a library of background knowledge (.pbl file specifying the domain); (2) a conceptual model (.pbm le specifying the problem); (3) a data file; and (4) a task specification .xml file, specifying the particular task.

[1] To this end, running ProBMoT require cumbersome and time-demanding procedures of preparing an appropriate input. For instance, the .xml task specification file defines all the hyper-parameters needed for ProBMoT to run properly, such as the paths of the input files, definition and mapping of variables and outputs to data sets, settings of the parameter fitter and the simulator, etc. All these components are represented with different XML tags. In response, the main contribution of ProBMoTViz is facilitating the task of process-based modeling with ProBMoT. In particular, the ProBMoT workflow will be encapsulated in a shell, where the .xml file is not written manually, but its representative settings are tuned interactively.

On the other hand, the constructed process-based models can be complex and difficult to understand[2]. The textual representation of process-based models can be improved, thus further enhancing their interpretability and communicability with domain experts. ProBMoTViz implements state-of-the-art visualization techniques able to overcome the potential comprehensibility obstacles and usability limitations of the current textual representations.

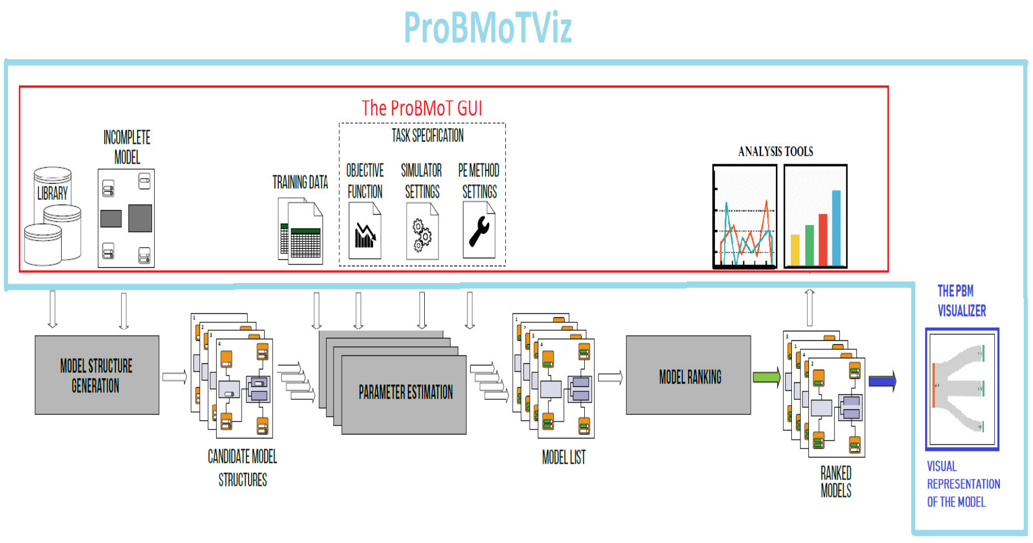


Figure 1. ProBMoTViz, the proposed extension of ProBMoT, consists of a GUI for ProBMoT and a PBM Visualizer

[1]Tables 3-5 in the Appendix, present these inputs for a particular modeling task.

[2] Table 6 in the Appendix

3. PROBMOTVIZ

ProBMoTViz consists of two main components: the ProB-MoT GUI and the PBM Visualizer (Figure 1). The former guides the user through the process of creating and defining a new PBM task step-by-step. It is a desktop-based application, developed in JavaFX [12], that facilitates the task of PBM, allowing for the settings, mappings, and all of the other customizable properties which must be specified in the .xml task settings file, to be now adjusted interactively in a workflow. The platform is divided into nine main scenes through which the user must progress in order to define the modeling task, monitor its progress, as well as to analyze the process-based models obtained at output. In particular, the scene sequence is as follows: (1) Library - the scene where the library file must be chosen and all the library components (template entities and processes) are shown; (2) Model - the scene where the (in)complete model file must be chosen and all the model components (instance entities and processes) are shown; (3) Data - the scene where the data les must be chosen, with the opportunity to inspect/ visualize the data; (4) Inputs - the scene where the input mappings must be specified, i.e., the mapping of the time dimension and the exogenous variables to a column in the data set; (5) Outputs - the scene where the outputs and their mappings to a specific column in the data set must be specified; (6) Overview scene; (7) Settings - the scene where all the task settings are specified, i.e., the evaluation, simulator, fitter, and other settings; (8) Run scene - where the particular task can be exported in an .xml format for later (re)use and (9) Results - the scene where the resulting process-based models can be inspected and analyzed.

The latter component, the PBM Visualizer, is a web application produced using the D3.js (Data Driven Documents JavaScript library) [2]. It offers an interactive visual representation of the complex hierarchies of process-based models depicting both the high-level structure of the models as well as the interactions between its components. In particular, process-based models are depicted as a Sankey diagram [1], where the nodes denote the components of the process-based models. We define two main types of components that correspond to: entities and processes. Moreover, entities have one sub-type: hierarchical entities (representing the hierarchy that the entity comes from), while the processes have two sub-types: hierarchical processes (representing the hierarchy that the process comes from), and children processes (representing the nested processes in a process), as shown in Figure 2.

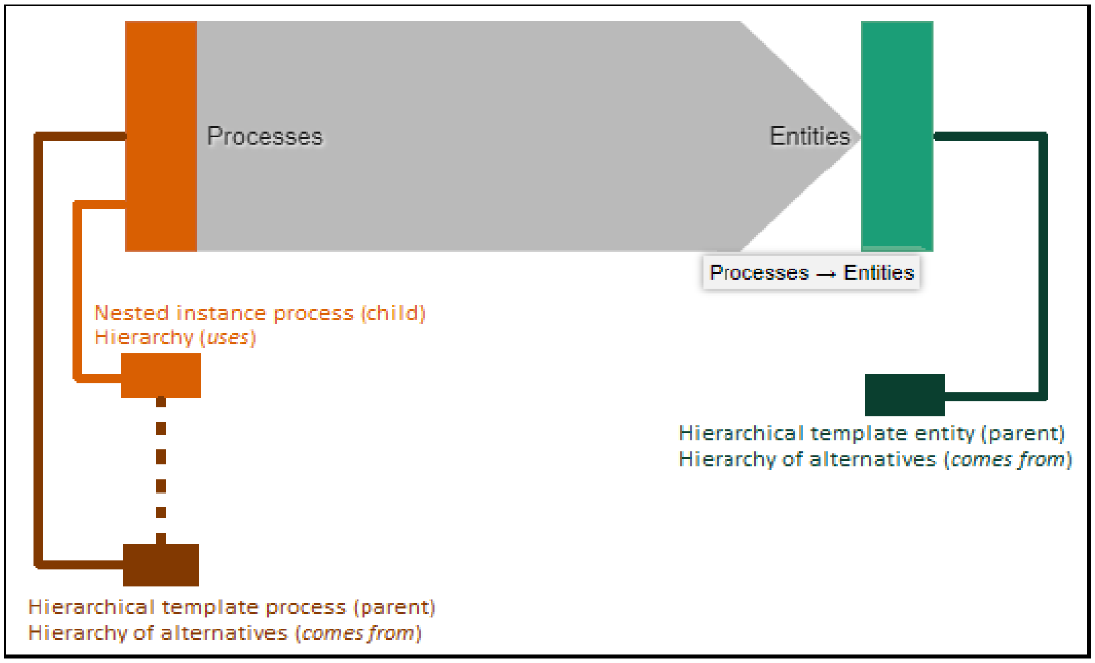
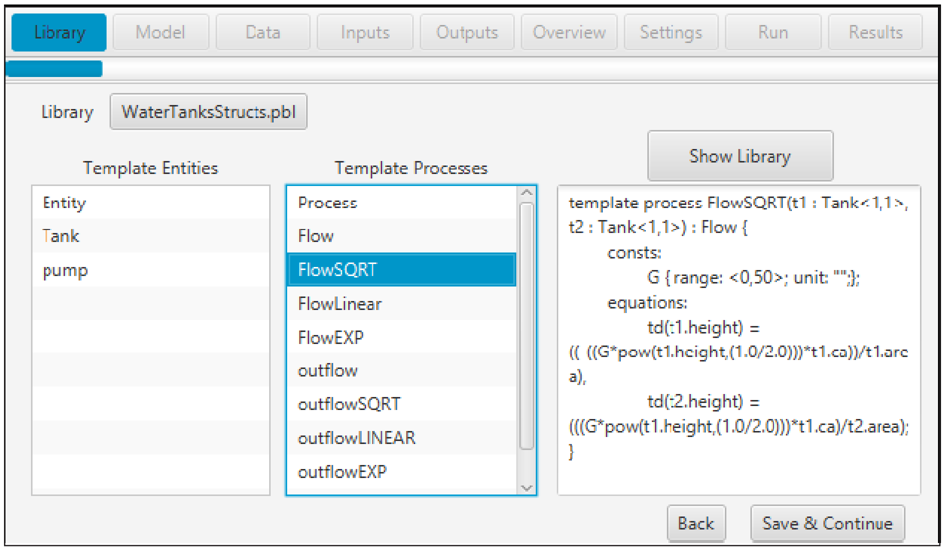

Processes
Entities
Processes → Entities
Nested instance process (child)
Hierarchy (*uses*)
Hierarchical template entity (parent)
Hierarchy of alternatives (*comes from*)
Hierarchical template process (parent)
Hierarchy of alternatives (*comes from*)

Figure 2. Schematics of the visual representation of model components

To better illustrate how ProBMoTViz works, we present the important details of preparing a task for modeling a two cascaded water tanks system (Equation 1). The system consists of two cascaded water tanks with free outlets, placed one above the other, fed by a pump. In the governing equations for this system, the water levels of the tanks are denoted with $h1$ and $h2$. $A1$, $A2$, $a1$ and a2 denote the areas of the tanks and their effluent areas, while the applied voltage-to- flow conversion constant is denoted with k. The task is to model the water level in the lower tank. The data is obtained by laboratory measurements [13] and it consists of 2500 samples (1500 train and 1000 test set) of the input voltage applied to the pump and the water levels in both tanks.

$$\begin{cases} \frac{dh_1}{dt} = -\frac{a_1\sqrt{2g}}{A_1}\sqrt{h_1} + \frac{k}{A_1}u(t) \\ \frac{dh_2}{dt} = -\frac{a_2\sqrt{2g}}{A_2}\sqrt{h_2} + \frac{a_1\sqrt{2g}}{A_2}\sqrt{h_1} \end{cases} \tag{1}$$

First, after loading the PBM library into ProBMoTViz (Figure 3), one can explore the encoded domain knowledge (for this example, for modeling fluid dynamics) in the traditional PBM formalism. In particular, the entity Tank encodes a variable that represents its water height level, and constants denoting the inflow and outflow areas. Analogously, the other entity Pump incorporates a variable denoting the input voltage in the system. Moreover, the library also encodes the different (plausible) interactions between the entities in terms of water transmissions between: two tanks, a tank and the environment as well as a tank and a pump. Note that, these interactions can also have different behavior, therefore the library encodes different modeling alternatives for each of them in terms of a squared-root, linear or exponential dynamics.



```
template process FlowSQRT(t1 : Tank<1,1>,
t2 : Tank<1,1>) : Flow {
        consts:
                G { range: <0,50>; unit: ""};
        equations:
                td(:1.height) =
(( ((G*pow(t1.hcight,(1.0/2.0)))*t1.ca))/t1.arc
a),
                td(:2.height) =
(((G*pow(t1.height,(1.0/2.0)))*t1.ca)/t2.area);
}
```

Figure 3. The water tanks library of background knowledge

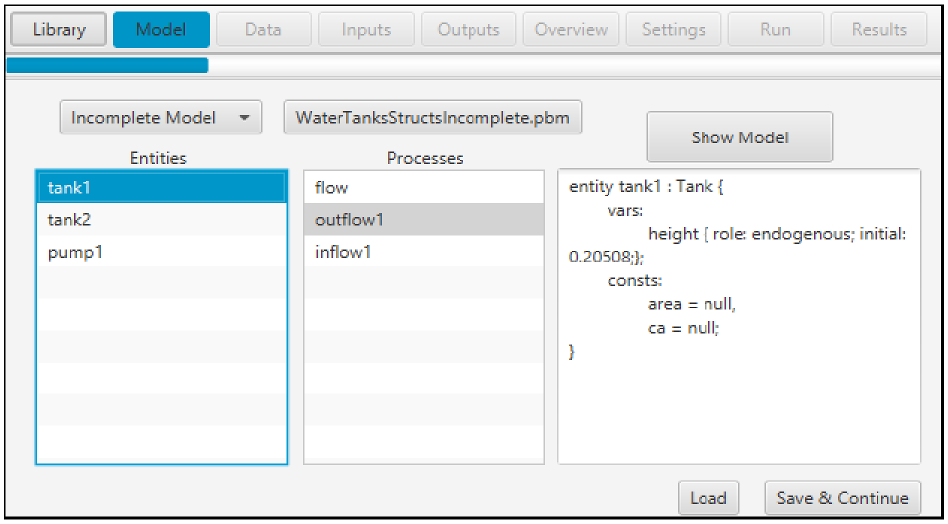
In the next step, after loading the incomplete model (Figure 4), one can explore the specific components of the particular problem, i.e. two tanks and one pump. The task is defined as follows: in a two-tank system with an electric pump, identify the underlying dynamics of the three different interactions that describes the behavior of the lower thank.

With loading the data, followed by specifying the mappings, defining the outputs and specifying the settings, our particular task is completely defined and ready for execution, as shown in Figure 5.

Finally, ProBMoTViz, lists the constructed models (Figure 6), and offers additional tools (error-plots and visual representation of the constructed process-based models) for further analyses.

4. Case Studies

As a case study, we tackle the tasks of modeling two benchmark nonlinear dynamical systems: (1) Two cascaded water tanks system and (2) The SilverBox { an oscillatory system using ProBMoTViz. For evaluating the performance of our models, we measure the relative root mean squared error (RRMSE) of each model's output, shown in Equation 2. The number of samples in the test set is denoted with n; y_t is the measured and $\hat{y}_t$ is the predicted value (obtained by simulating the model m on the te-



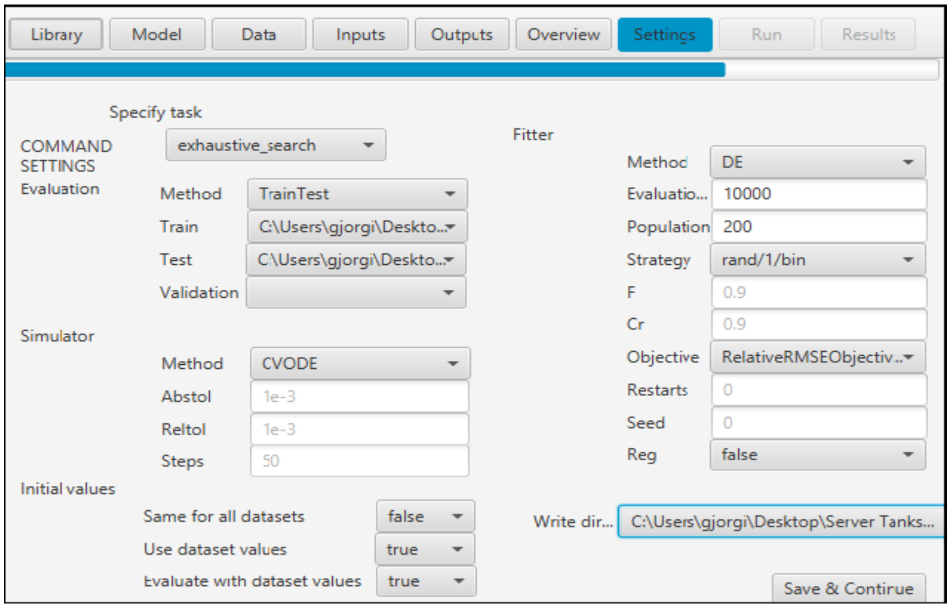
Incomplete Model
WaterTanksStructsIncomplete.pbm

Entities: tank1, tank2, pump1

Processes: flow, outflow1, inflow1

```
entity tank1 : Tank {
        vars:
                height { role: endogenous; initial:
0.20508;};
        consts:
                area = null,
                ca = null;
}
```

Figure 4. The water tanks conceptual model



Specify task

COMMAND SETTINGS: exhaustive_search

Evaluation
- Method: TrainTest
- Train: C:\Users\gjorgi\Deskto...
- Test: C:\Users\gjorgi\Deskto...
- Validation:

Simulator
- Method: CVODE
- Abstol: 1e-3
- Reltol: 1e-3
- Steps: 50

Initial values
- Same for all datasets: false
- Use dataset values: true
- Evaluate with dataset values: true

Fitter
- Method: DE
- Evaluatio...: 10000
- Population: 200
- Strategy: rand/1/bin
- F: 0.9
- Cr: 0.9
- Objective: RelativeRMSEObjectiv...
- Restarts: 0
- Seed: 0
- Reg: false

Write dir...: C:\Users\gjorgi\Desktop\Server Tanks...

Figure 5. The settings parameters for modeling the two water tanks system shown

st set) of the system variable y at time point t. The mean value of y in the test set is denoted with y. This metric is relative to the standard deviation of the system variable in the test data, thus allowing us to compare the errors of models for different system variables with measured values on different scales.

$$RRMSE(m) = \sqrt{\frac{\sum_{t=0}^{n}(y_t - \hat{y}_t)^2}{\sum_{t=0}^{n}(y_t - \bar{y})^2}} \qquad (2)$$

4.1 Two Cascaded Water Tanks System

For the previously defined water tanks system, given an input voltage, the output of interest in our model is the water height level of the lower tank. The particular process-based modeling task yields 9 feasible models, as shown in Table 1. The best obtained model (Figure 7) is contained of the processes Inflow, FlowSQRT and OutflowSQRT, which corresponds to the original system.

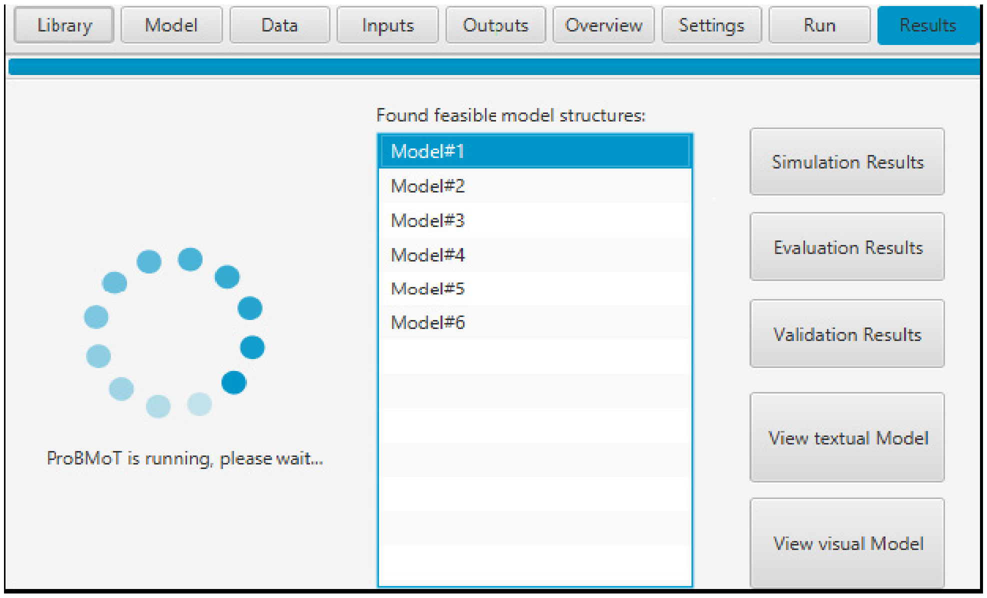


Figure 6. The resulting models from modeling the two water tanks system

Model	**Train RRMSE**	**Test RRMSE**
ModelSqrtSqrt	0.2208	0.2673
ModelLinSqrt	0.2285	1.1837
ModelExpSqrt	0.2448	0.3101
ModelSqrtLin	0.2916	0.3200
ModelLinLin	0.3138	0.3323
ModelExpLin	0.3576	0.4249
ModelSqrtExp	0.7233	0.8507
ModelLinExp	0.7312	0.8585
ModelExpExp	0.7457	0.8835

Table 1. The results of modeling the two cascaded water tanks system

4.2 SilverBox Oscillator System

The second case study, addresses the task of reconstructing a nonlinear mechanical oscillating system, referred as the Silver Box system - an electronic implementation of the During oscillator. The system's dynamics (Equation 3) relates to the displacement y (t) (the output) to the input voltage $u(t)$. The parameter m is a moving mass, d is viscous damping, and k (y) is a nonlinear progressive spring described by a static but position-dependent stiffness. The data is generated by an almost idealized representation of the oscillator [13]. It consists of 130000 samples (90000 train 40000 test set) of the input voltage and the output displacement of the oscillator.

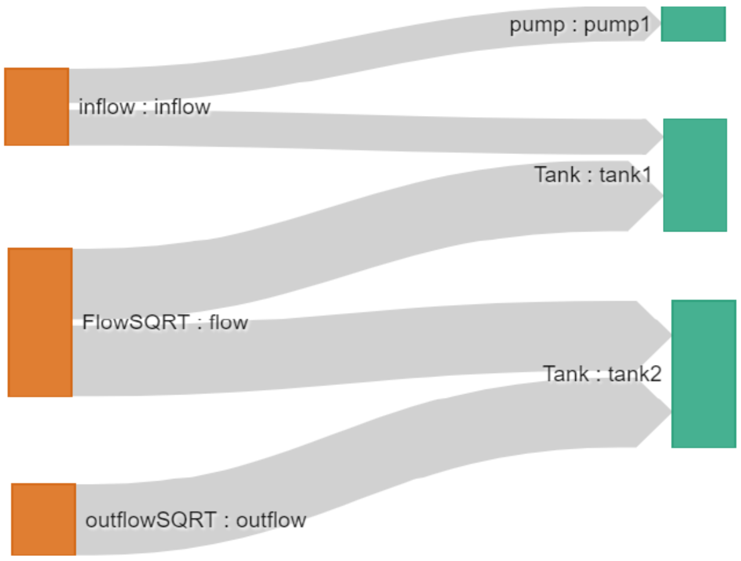


Figure 7. The water tanks visual process-based model

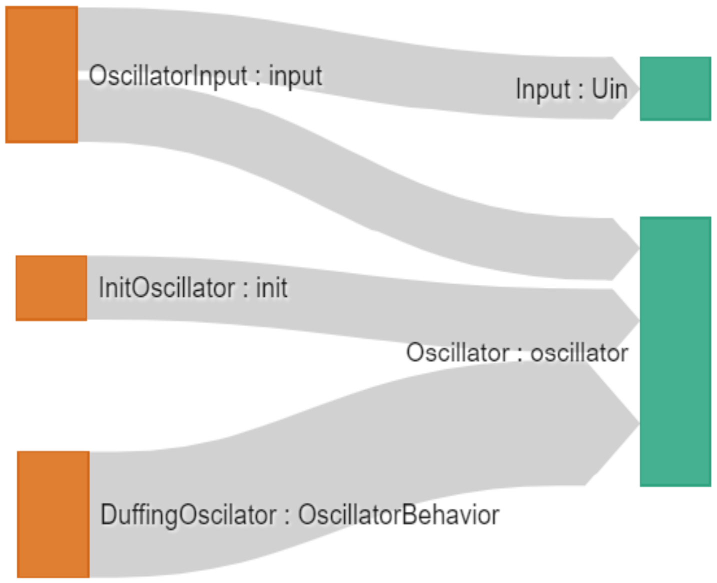


Figure 8. The SilverBox visual process-based model

$$\begin{cases} m\frac{d^2y(t)}{dt^2} + d\frac{dy(t)}{dt} + k(y(t))y(t) = u(t) \\ k(y(t)) = a + by^2(t) \end{cases} \tag{3}$$

The PBM library incorporates the domain knowledge where the entity Oscillator encodes the input voltage, output displacement and it’s mass. Moreover, behaviors of different oscillators: (1) Dung, (2) Simple, (3) Harmonic and (4) Universal oscillator, all of which differently affect the output displacement of the oscillator are also encoded in the library. The incomplete model species the particular problem of one oscillator with unknown oscillatory behavior.

The process-based modeling task yields 4 feasible process based models, as shown in Table 2. The best obtained model (Figure 8) contains the processes Init Oscillator, Oscillator- Input and Dung Oscillator, which corresponds to the original system.

Model	Train RRMSE	Test RRMSE
ModelDung	0.2353	0.2741
ModelSimple	∞	∞
ModelUniversal	0.8803	0.8796
ModelHarmonic	0.9330	0.9327

Table 2. The results of modeling the SilverBox system

5. Conclusions

In this work, we present a novel software tool, ProBMoTViz. It consists of two components: a GUI for ProBMoT and a PBM Visualizer for process-based models. The GUI supports the basic operations necessary for (automated) modeling of dynamical systems. Moreover, it allows for comprehensible and user-friendly preparation and analyses of modeling tasks. This enables the user to have better overview and control over the input parameters necessary when running ProBMoT, therefore saving time and computational resources when performing large amount of- and/or delicate experiments. The PBM Visualizer, on the other hand, aids in visualizing the complex (hierarchical) structure of process-based models, in turn allowing for better understanding and comprehensibility.

6. References

[1] Atkinson, N. (2015). Bi-directional hierarchical sankey diagram [online: https://github.com/neilos/bihisankey], 2015.

[2] Bostock, M., Ogievetsky, V., Heer, J. (2011). D3 data-driven documents. IEEE transactions on visualization and computer graphics, 17(12):2301-2309.

[3] Bridewell, W., Langley, P., Todorovski, L., Dzeroski, S. (2008). Inductive process modeling. Machine learning, 71(1):1-32.

[4] D. —Cerepnalkoski. Process-based Models of Dynamical Systems: Representation and Induction: Doctoral Dissertation. PhD thesis, D. Cerepnalkoski, 2013.

[5] D—zeroski, S., Todorovski, L. .(2002). Encoding and using domain knowledge on population dynamics for equation discovery. In Logical and computational aspects of model-based reasoning, p 227-247. Springer, 2002.

[6] Langley, P., Sanchez, J. N., Todo111rovski, L., Dzeroski, S. (2002). Inducing process models from continuous data.

[7] Peev, G., Simidjievski, N., Dzeroski. S. (2017). Modeling of dynamical systems: a survey of tools and a case study. In 20th *International Multiconference Information Society* - IS 2017, A, *p*15-18.

[8] Peev, G., Simidjievski, N., Dzeroski, S. (2018). Identification of a nonlinear dynamical benchmark system using process-based modeling. In 10th JoÅ¦¿ef Stefan IPS Students' Conference, p 36, 2018.

[9] Simidjievski, N., Todorovski, L., Dzeroski, S. (2015). Predicting long-term population dynamics with bagging and boosting of process-based models. *Expert Systems with Applications*, 42 (22), 8484-8496, 2015.

[10] Tanevski, J., Simidjievski, N., Todorovski, L., Dzeroski, S. (2017). Process-based modeling and design of dynamical systems. In ECML PKDD, p 378-382. Springer, 2017.

[11] Tanevski, J., Todorovski, L., Dzeroski, S. (2016). Process-based design of dynamical biological systems. *Scientic reports*, 6:34107.

[12] Topley. K. (2010). JavaFX Developer's Guide. Pearson Education, 2010.

[13] Wigren, T., Schoukens, J. (2013). Three free data sets for development and benchmarking in nonlinear system identifi cation. In 2013 *European Control Conference* (ECC), p 2933-2938, July 2013.

Appendix: ProBMoT inputs

Tables 3-5 present the necessary inputs for ProBMoT for the tasks of automated modeling of a water tank dynamic system. Table 6 presents a resulting process-based model in the standard PBM formalism.

```
library WaterTanksLibrary;
//ENTITIES
template entity Tank {
vars:
        height {aggregation:sum, range:<0,500>};
consts:
        area frange: <1.0E-3,30>},
        ca frange: <1.0E-3,30>}; }
        template entity Pump f
vars:
        v{aggregation:sum, range:<-15,15>};
consts:
        k {range:<0.2,1E6>}; }
//PROCESSES
template process Flow (t1 : Tank, t2 : Tank) {
consts:
        G {range: <0,50>},
template process FlowSQRT : Flow {
equations:
        td(t1.height) = - (G * pow(t1.height,1/2) * t1.ca)/t1.area,
        td(t2.height) = (G * pow(t1.height,1/2) * t1.ca)/t2.area; }
template process FlowLINEAR: Flow {
equations:
        td(t1.height) = - (G * t1.height * t1.ca)/t1.area,
        td(t2.height) = (G * t1.height * t1.ca)/t2.area; }
template process FlowEXP : Flow {
equations:
        td(t1.height) = - (G * exp(t1.height) * t1.ca)/t1.area,
        td(t2.height) = (G * exp(t1.height) * t1.ca)/t2.area; }
template process Outflow (t:Tank) {
consts:
        G {range: <0,50>}; }
        template process OutflowSQRT: Outflow {
equations:
        td(t.height) = - G * pow(t.height,1/2) * t.ca/t.area; }
        template process OutflowLINEAR :Outflow {
equations:
        td(t.height) = - G * t.height * t.ca/t.area; }
        template process OutflowEXP :Outflow {
equations:
        td(t.height) = - G * exp(t.height) * t.ca/t.area; }
        template process Inflow (p: pump, t: Tank) {
equations:
        td(t.height) = p.k * p.v/t.area; }
```

Table 3. Library of domain knowledge

```
incomplete model WaterTanksIncomplete : WaterTanksLibrary;
//Entities
entity tank1 : Tank {
vars:
        height { role: endogenous; initial: 0.20508;};
consts:
        area = null,
        ca = null;
}
entity tank2 : Tank {
vars:
        height { role: endogenous; initial: 0.38086;};
consts:
        area = null,
        ca = null;
}
entity pump1 : Pump {
vars:
        v { role: exogenous;};
consts:
        k = null;
}
//Processes
process flow (tank1, tank2) : Flow {
consts:
        G = 4.429; }
process outflow1 (tank2) : Outflow {
consts:
        G = 4.429; }
process in ow1 (pump1, tank1): Inflow {
}
```

Table 4. Incomplete model of the two cascaded water tanks system

```
<task>
        <library>C:/Users/WaterTanksLibrary.pbl</library>
        <incomplete>C:/Users/WaterTanksIncomplete.pbm</incomplete>
        <data>
                <d separator="," id="1">C:/Users/Data1.csv</d>
                <d separator="," id="2">C:/Users/Data2.csv</d>
</data>
<mappings>
```

```
<dimensions>
            <dim name="time" col="t"/>
      </dimensions>
      <exogenous>
<exo name="WaterTanksIncomplete.pump1.v" col="u"/> </exogenous>
<endogenous>
      <endo name="WaterTanksIncomplete.tank1.height" col = "h1 "/>
      <endo name="WaterTanksIncomplete.tank2.height" col = "h2 "/>
</endogenous>
      <outputs>
            <out name = "h2 " col = "h2 "/>
      </outputs>
</mappings>
<output>
      <variables>
            <var name="h2">WaterTanksIncomplete.tank2.height</var>
      </variables>
</output>
<writeDir>C:/Users/  </writeDir>
<command>exhaustive search</command>
<settings>
      <initialvalues>
            <sameforalldatasets>false</sameforalldatasets>
            <usedatasetvalues>true</usedatasetvalues>
</initialvalues>
<simulator method="CVODE">
      <abstol>0.001</abstol>
      <reltol>0.001</reltol>
      <steps>1000</steps>
</simulator>
<fitter method="DE">
            <restarts>0</restarts>
            <evaluations>10000</evaluations>
            <population>200</population>
            <strategy>rand/1/bin</strategy>
            <F>0.9</F>
            <Cr>0.9</Cr>
            <seed>0</seed>
            <reg>false</reg>
            <object ives>
                  <obj>RelativeRMSEObjectiveFunctionMultiDataset</obj>
            </objectives>
      </fitter>
      <evaluation method="TrainTest">
                  <train>1</train>
                  <test>2</test>
            </evaluation>
      </settings>
</task>
```

Table 5. An example task specication le in XML format

```
model WaterTanksModel : WaterTanksLibrary;
entity tank1 : Tank {
vars:
        height { role: endogenous; initial: 0.20508;};
consts:
        area = 19.944,
        ca = 1.087;
}
entity tank2 : Tank {
vars:
        height { role: endogenous; initial: 0.38086; };
consts:
        area = 23.051,
        ca = 3.066;
}
entity pump1 : Pump {
vars:
        v { role: exogenous;};
consts:
        k = 20.305;
}
//Processes
processflow (tank1, tank2) : FlowSQRT{
consts:
        G = 4.429; }
process out flow1 (tank2) : Out flowSQRT {
consts:
        G = 4.429; }
process in flow1 (pump1, tank1): Inflow {
}
```

Table 6. A process-based model of a two cascaded water tanks system