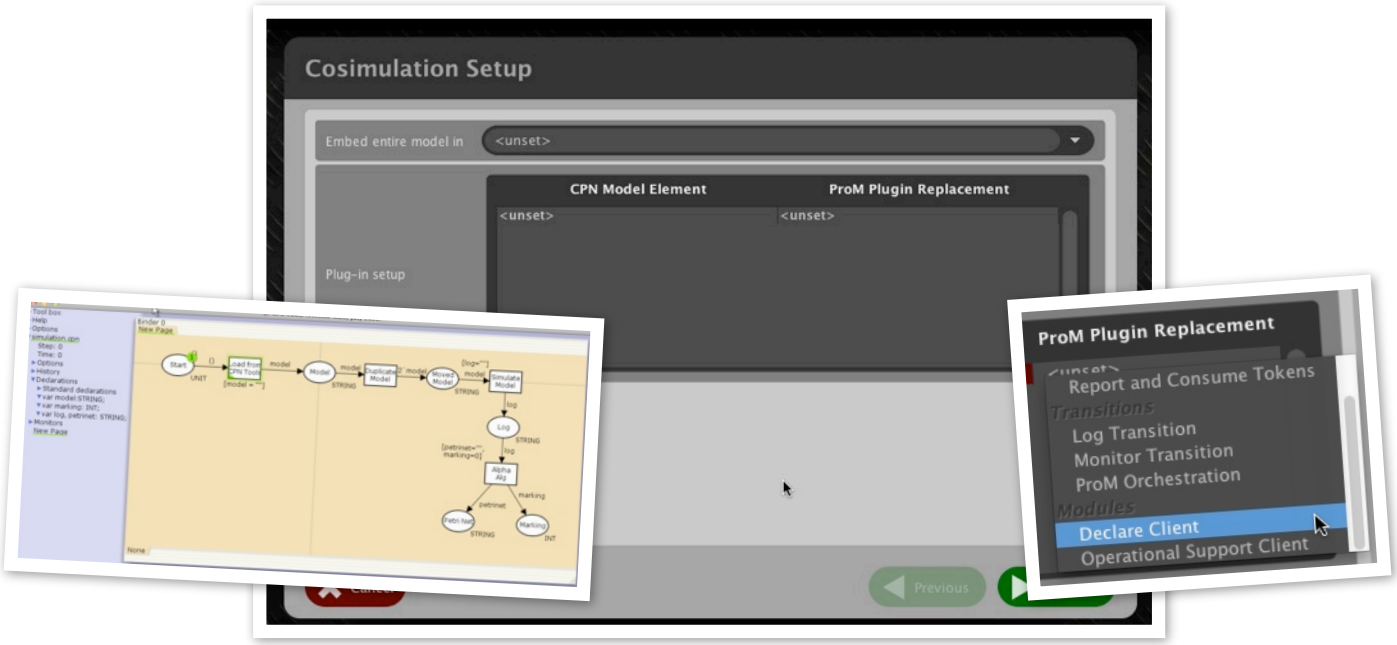


Cosimulation

Michael Westergaard

Wednesday, July 13, 2011

The Cosimulation plug-in incorporates the Access/CPN 2.0 feature to run a CPN model in parallel with Java code. The plug-in provides a user interface on top of Access/CPN 2.0 and a few simple sample plug-ins.

This document assumes that readers are familiar with the CPNet documentation (see http://westergaard.eu/2011/07/prom-package-documentation-cpnet/ for more information). It may also be beneficial to be familiar with Access/CPN 2.0, described in

> M. Westergaard
> **Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models**
> Proceedings of 32nd International Conference on Application and Theory of Petri Nets. Newcastle, England, June 2011.

which you can find at http://www.springerlink.com/content/45401t4j223x5715/.

Here, we explain how to set up a cosimulation and the various plug-ins included in this package.

External References

Simple demo of cosimulation and ProM orchestration: http://westergaard.eu/2010/07/cosimulating-cpn-models-and-prom-plugins-and-application-to-prom-orchestration/

More advanced demo of cosimulation using subpage plug-ins: http://westergaard.eu/2010/07/embedding-declare-in-cpn-models-via-prom/

Suggestion for extension of ProM orchestration plug-in mentioned later: http://westergaard.eu/2011/05/prom-plug-in-suggestion-prom-orchestration/

Presentation explaining the background for this work: http://westergaard.eu/2011/05/using-declarative-workflow-modeling-to-achieve-happiness-in-life/

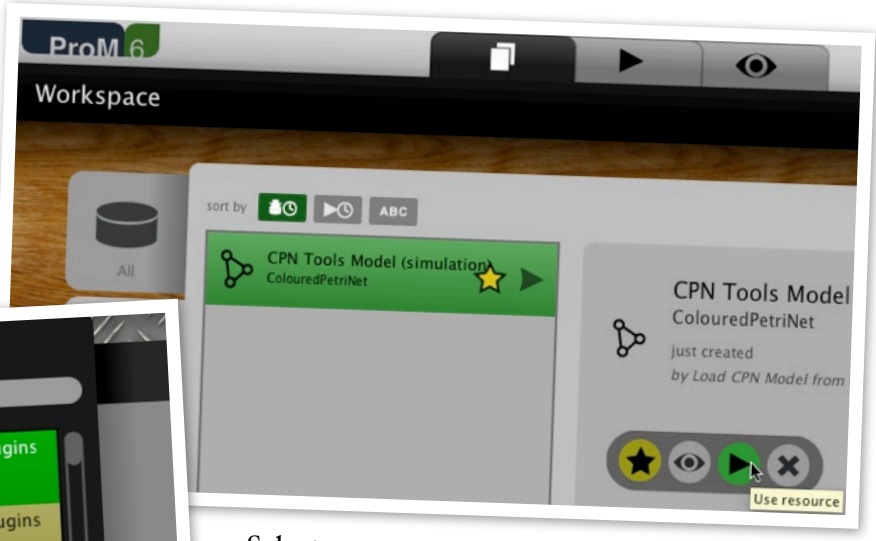
Paper describing Access/CPN 2.0: http://www.springerlink.com/content/45401t4j223x5715/

Presentation of above paper: http://westergaard.eu/?p=2891

Setting Up and Running Cosimulation

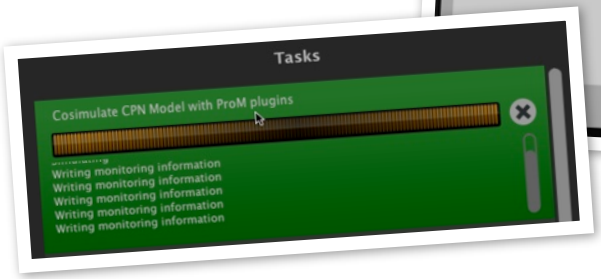
Setting up and running a cosimulation is done using these steps:

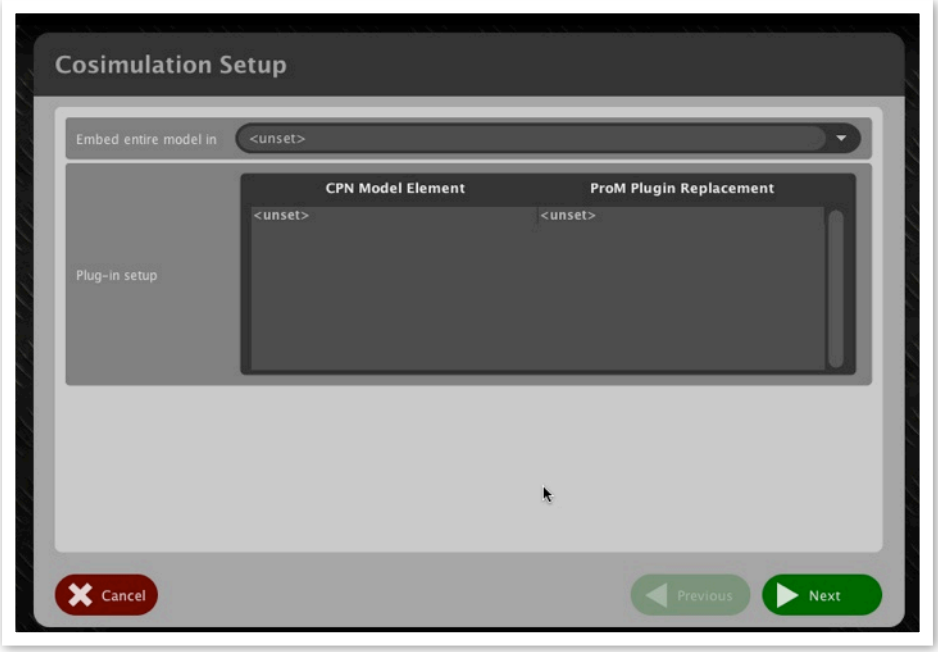
1. Load a CPN model as explained in the CPNet documentation.
2. Select the loaded model in the Workspace and click on Use resource (the thing that looks more like an arrow than like a battleship).



3. Select the Cosimulate CPN Model with ProM plugins plug-in. You have a choice among two options: one taking just a CPN model and one additionally taking a start marking. As no plug-in currently produces markings, going for the one without is a winning option.
4. You now get the configuration screen for setting up your cosimulation. We explain this dialog in the following subsection. Fill it in and press Next.
5. Next, you have to set up the cosimulation. The options for this are the same as for setting up a normal simulation (except you cannot do repetition) as explained in the documentation for CPNet.

6. Now, either the cosimulation starts or you get configuration screens for the individual plug-ins you use.





Configuration Dialog

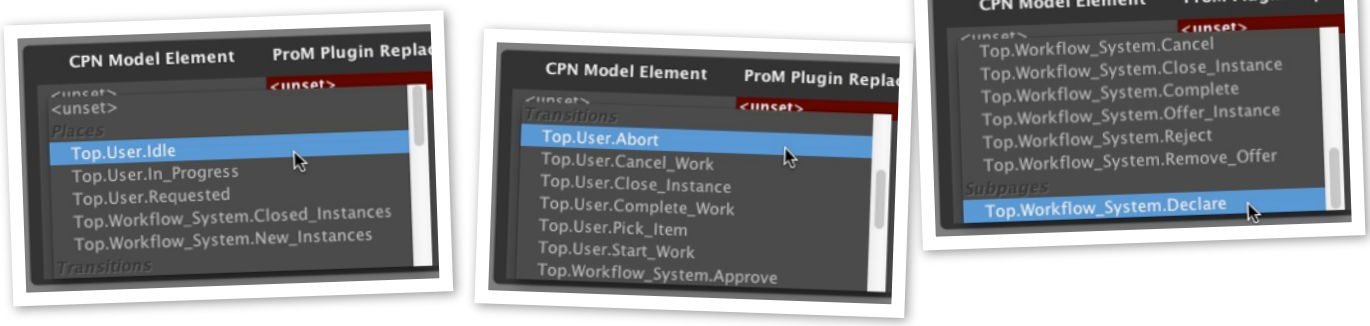
Configuration comprises two parts: setting an environment for the model and attaching plug-ins to net elements.

To set an environment do this:

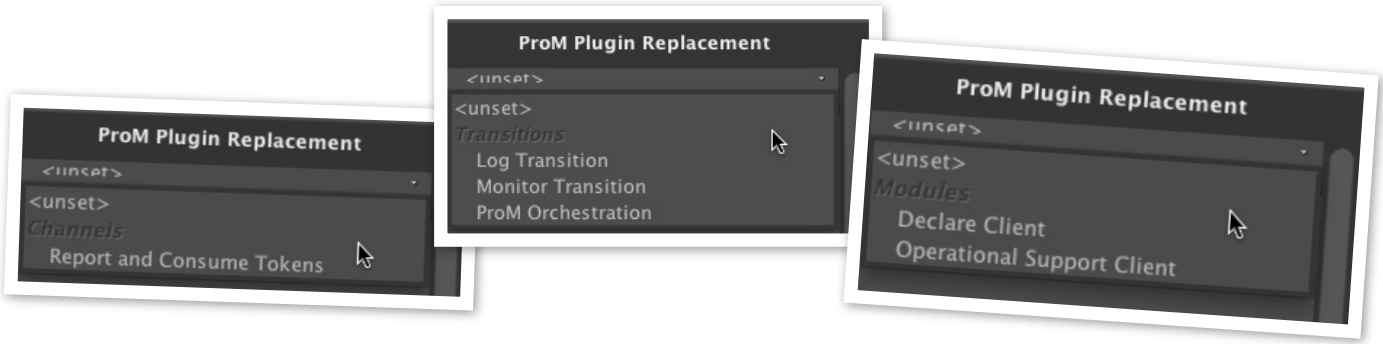
1. Click on the drop-down Embed entire model in.
2. Select the environment you wish to embed the entire model in.

To replace a net element by a plug-in, you have two possibilities: you can either first select a model-element and then an appropriate replacement or you can select the replacement first. To use the first approach do this:

1. Select your favorite net element for replacement. You can pick places, transitions, or sub-pages. Pick an element by clicking on an <unset> cell under the CPN Model Element header. New rows are added automatically.

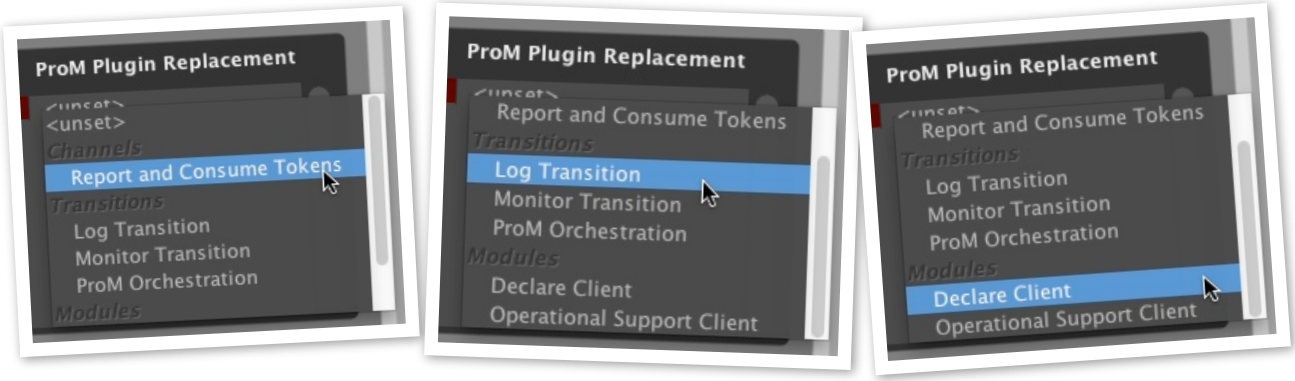


2. Select your desired replacement by clicking on <unset> under ProM Plugin Replacement in the same row as the desired net element. Only matching replacements will be shown.

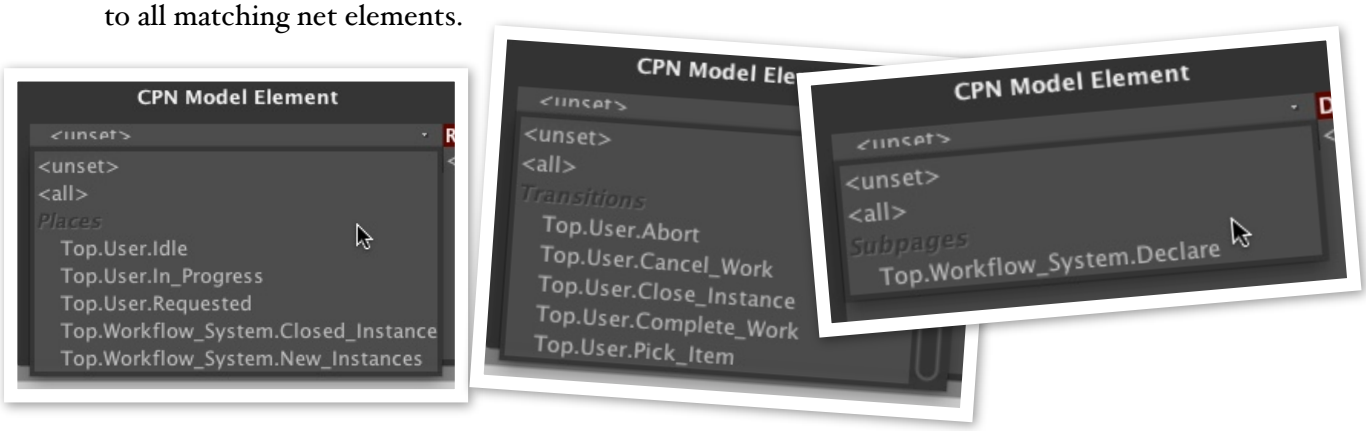


You can also tie a net element to a plug-in by first selecting the replacement as follows:

1. Select a replacement by clicking on <unset> under the ProM Plugin Replacement column.

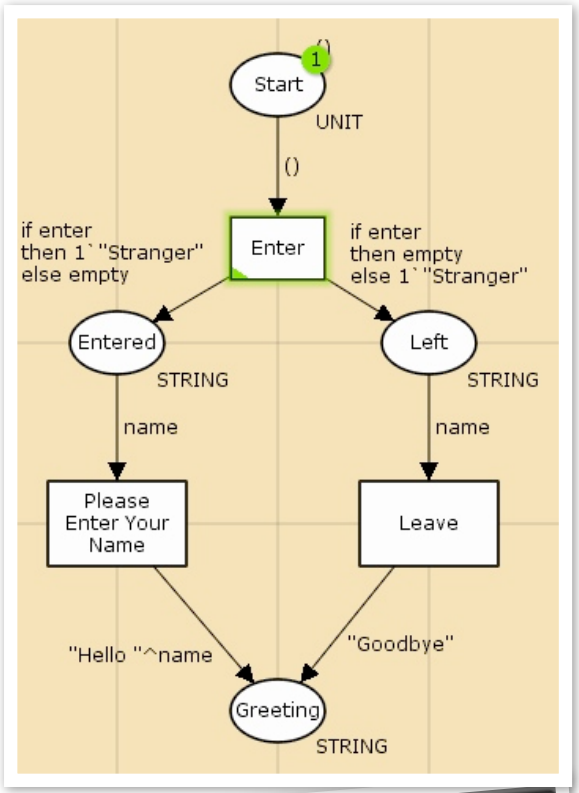


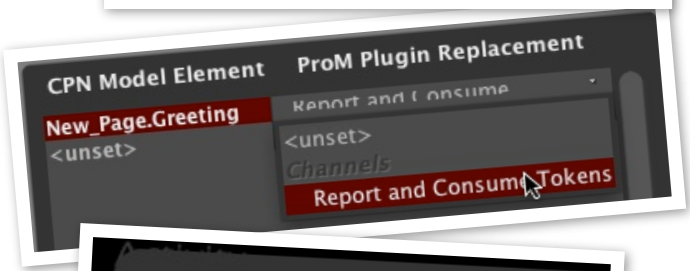
2. Select the net element you want to replace. Only relevant net elements are shown. In addition, you can select <all>, which attaches the plug-in to all matching net elements.

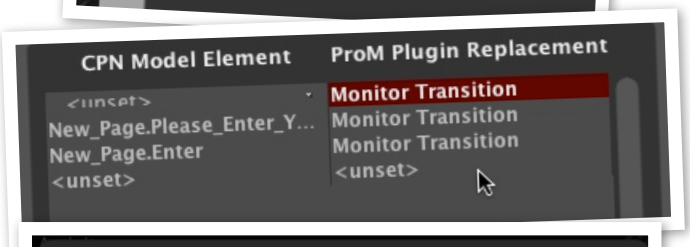


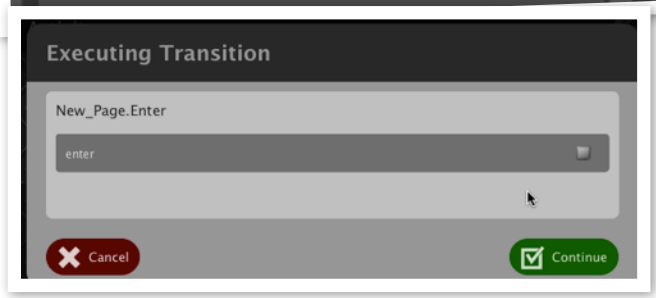
Note: If either the net element or the plug-in replacement is set to <unset>, the line is ignored when running the cosimulation.

Hint: If you want to attach the same plug-in replacement to all or nearly all net elements of the correct type (this is very useful for, e.g., ProM Orchestration presented later), an efficient means of doing so is to first select the replacement, then select <all> and finally remove all net elements you are not interested in.









Simple Plugin Replacements

This package ships with a number of demonstration plug-ins. In this section, we describe the 3 simplest such plug-ins, which are little more than tests and technology demos. They may be useful for debugging purposes, though, so here we focus on the end-user perspective. For the programmer, we mention the class implementing each of them. The implementations are in all cases simple enough that with a little effort, it can be understood. In the next section, we describe the more advanced ProM Orchestration replacement, which is more advanced and more useful.

In the following we use the example to the left (or right? *this* hand in any case). We can think of it as a concierge at a hotel. A person either enters or looks at the hotel and leaves. If the person enters, he is asked for his name and subsequently greeted. If the person doesn't enter, the concierge says goodbye.

Report and Consume Tokens

Place plug-in: This plug-in makes it easy to show results of simulation. Whenever a token is produced on the associated place, it is consumed and the value is shown in the ProM GUI.

Attaching this plug-in to the Greeting place of our example model, can yield a greeting like the one to the left.

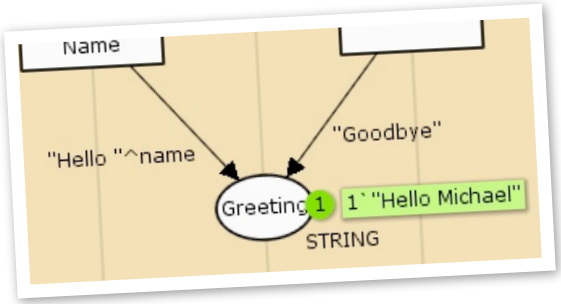
Monitor Transition

Transition plug-in: This plug-in notifies a user whenever an attached transition is executed. The executed binding can even be modified or execution can be aborted.

Attaching the transition to the Enter and Please Enter Your Name transitions allows a user to decide whether the guest enters or leaves the hotel (left). The boolean variable is displayed as a checkbox, and a user can check or uncheck it and

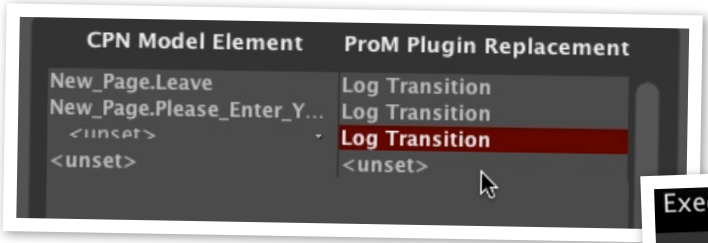
click Continue to execute the transition is the desired binding. Note that the transition is not exactly executed in the binding indicated, but the pre-set of the transition is executed in the original binding chosen by the simulator, and the post-set in the modified binding. The details are explained in http://www.springerlink.com/content/45401t4j223x5715/ on page 333. A user can also press Cancel to abort execution of the transition.

When executing Please Enter Your Name with the plug-in attached, we get the screenshot to the right, and can enter our name. Here, rebinding means, we consume the token "Stranger" from Entered and produce a personalized greeting on Greeting (which, by the way can be displayed in ProM using the Report and Consume Tokens plug-in).



Log Transition

Transition plug-in: This plug-in simply adds a transition to a log. The main purpose of this plug-in is to show how to allow multiple instances of the same plug-in to communicate, but it can be useful for logging and execution for further processing.

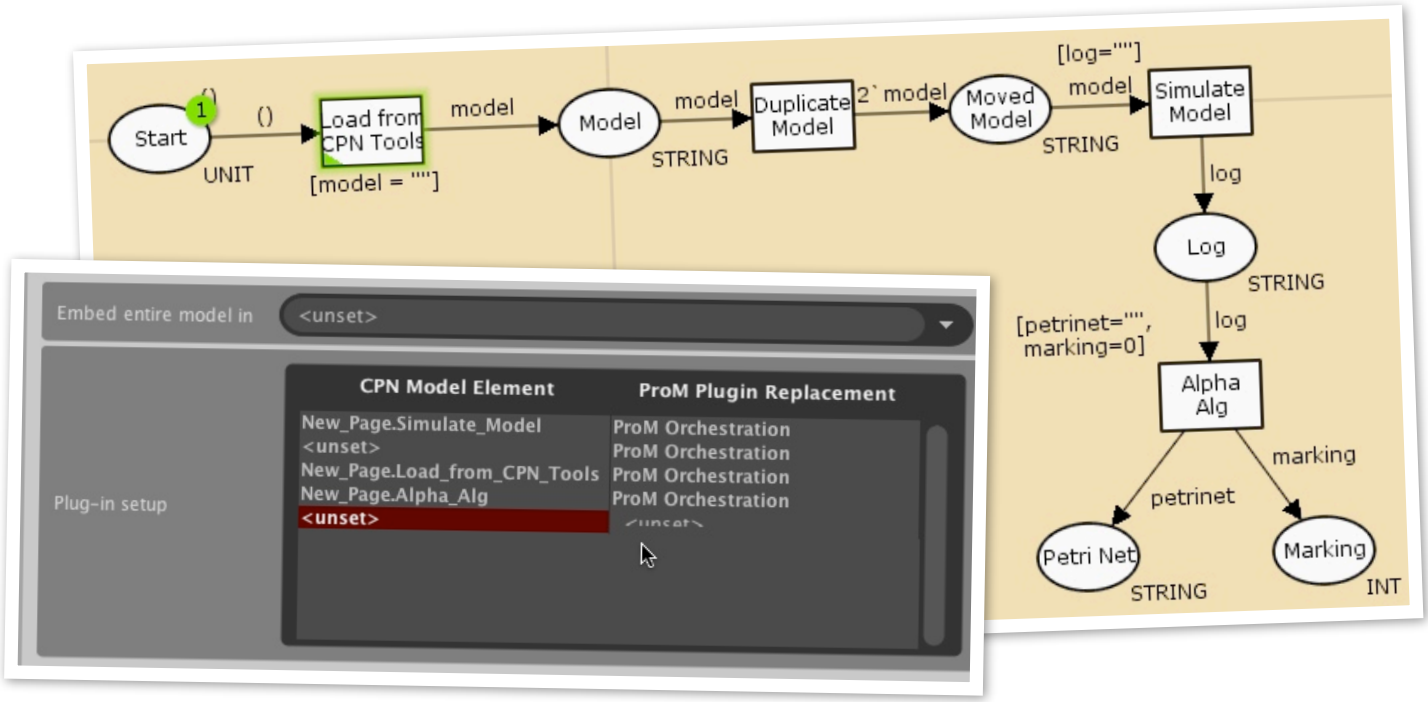


In our example, we attach it to the Leave and Please Enter You Name transitions and obtain the log below.

ProM Orchestration

Transition plug-in: This plug-in was made as a demonstration as well, but has proved to be very versatile and useful, and shall hence get a bit more thorough explanation.

The basic idea is to associate each transition in a CPN model with a plug-in in ProM. The matching is done completely automatically. Future versions may allow more user control, but the



current version for simplicity allows no user-control of the mapping between transitions and plug-ins. Tokens are associated with provided objects in ProM (the objects in the Workspace), and places contain tokens. Tokens can be moved freely and the places have no influence on the mapping between tokens and actual objects. Just like the mapping between transitions and plug-ins is done automatically, so is the mapping between tokens and objects.

Consider the example above. This describes a simple workflow to load a model, do simulations and finally do process mining on the simulation logs. To see the quality of the mined model, we run process mining twice on two separate logs generated from the same model.

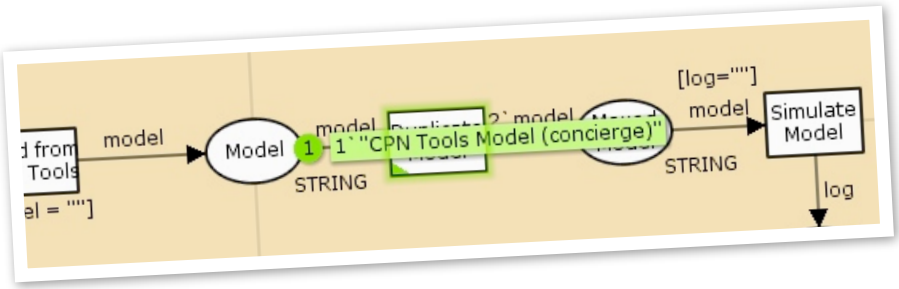
All tasks in the model, except for Duplicate Model, correspond to a plug-in in ProM. The names are not exactly the same, just similar, and the edit distance between all plug-ins and the transition name is used to pick the appropriate plug-in. Furthermore, the input and output values of the plug-in ProM is used in the matching.

Only places of types string, integer, and index are considered when computing input values. Thus, we can use a place of type unit (Start) to ensure the process executes only once.

Input Matching

Input values are matched as any expression on an input arc that comprises exactly one variable that is bound to something that has earlier been produced as output of another plug-in. Thus, when Simulate Model is executed, model is considered an input as the arc from Moved Model has an arc expression that is just a value and the token will be equal to the value earlier produced by Load from CPN Tools. Objects are matched based on the name of the place it originates from and the type of the original object.

Simple values (integers, booleans, and strings) are also recognized as inputs if they do not correspond to a previously produced object.



Finally, expressions are considered as objects if they correspond exactly to a previously produced object. This should not be relied upon to construct values yourself to stay independent of the internal representation of tokens.

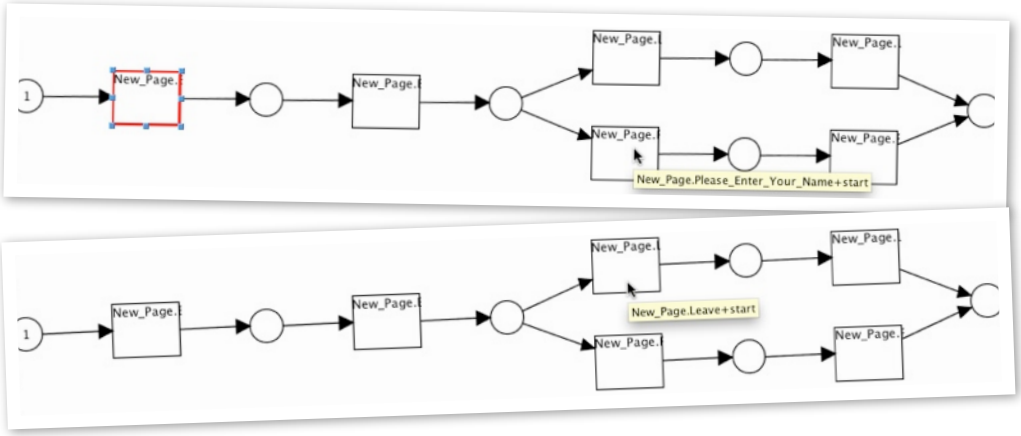
Plug-in Matching

As soon as all potential input values have been determined, all plug-ins matching these types are picked and the best one based on the name is chosen. If two plug-ins that are equally good matches exist, the one with the most specific PluginContext is chosen.

Output Generation

When a plug-in has been executed, values are produced and added to places with names similar to the tokens. The value of the tokens are computed automatically, using the name of the produced object if the resulting place has type string. At the top left of this page is a fragment of the model from earlier being executed.

Hint: Use type string on places to make it easier to debug your execution (as the tokens then are recognizable as objects in ProM).



Common pitfall: While simple types are understood as inputs, they are not understood as outputs. Hence, if a plug-in produces a string or integer, the value of the string/integer is not reflected in the CPN model, but instead a reference is generated which may have different value.

Example

Running the cosimulation overlaid on the model on the previous page with the model, and loading the model on page 5 for the loading step of the model, yields two logs (left of the previous page) and two resulting models (bottom left of the previous page). Both models allow first starting and completing the Enter task and then branches off to either leaving or asking for the name.

Known Limitations

- Only one place/transition plug-in should be attached to each place/transition, but the configuration does not enforce this.
- Simple types are handled in a non-optimal way in ProM orchestration. Check http://westergaard.eu/2011/05/prom-plug-in-suggestion-prom-orchestration/ for an in-depth explanation and suggestions for improvement.