

PIPE Experimenter User Guide

Marc Melià Aguiló, Catalina M. Lladó

July 2, 2008

1 Introduction

XML-based interchange formats for performance models provide a mechanism whereby performance model information may be transferred among modeling tools. This makes it possible for a user to create a model in one tool, perform some studies, and then move the model to another tool for other studies that are better done in the second tool. These formats specify the model and a set of parameters for one run. For model studies, however, it is useful to be able to specify multiple runs, or experiments, for the model.

In [4] an XML interchange schema extension, called Experiment Schema Extension (Ex-SE), defines a set of model runs and the output desired from them. This extension to an interchange schema provides a means of specifying performance studies that is independent of a given tool paradigm.

PN-Ex [3] is a specific extended instantiation of Ex-SE for Petri nets. An Experimenter for the Petri net tool PIPE2 (Platform Independent Petri net Editor 2) [2, 1], has been developed based on the PN-Ex. This document presents an overview of the PN-Ex format in Section 2 and the User Guide for the PIPE2 Experimenter in Section 3.

2 The PN-Ex format

In PN-Ex, an experiment is a specification of a set of runs to solve a Petri net (PN) with different parameters for each run.

For example, one may want to know how a system improves as the execution rate of a task is increased. In that case, an experiment with an iteration, with this task's rate as the parameter over which iterate would be specified. Moreover, more than one parameter can be changed at each run.

A PN-Ex file has two parts: the variable declaration part (not mandatory) and the execution part. There are 3 types of variables:

- **Variables:** these variables are linked by the experiment designer to a property of a PN element (for example initial marking of place, transition or arc weight). When the value of a variable changes, the corresponding PN element changes. These variables are used to change the parameters value in between runs in the experiment specification.
- **Output variables:** these variables are linked to the output of the previous experiment. For example, transition throughput, which makes it possible to stop an experiment when the throughput of a transition reaches a certain value.
- **Local variables:** local variables are not linked to any result or property. They just contain an expression. They are as the *variables* in most programming languages.

The execution part is composed by one or more Solution Specification (SolutionSpecs). They are executed following the same order they are specified in the PN-Ex file. At the beginning of each SolutionSpec, variables and output variables are reset to their initial values. Local variables keep their last value. Each SolutionSpec contains an Output Specification (OutputSpec) and a sequence of execution blocks.

The OutputSpec specifies the output in the scope of its SolutionSpec that has to appear in the results file. This results can be the content of variables and the output of calculations such as transition throughput or place utilization.

There are 4 types of execution blocks:

- **Assign:** to assign a value to a variable.
- **Solve:** when the experiment execution reaches a solve element the net is solved. There are different solution types which are described in the next section.
- **Iteration:** an iteration is a sequence of execution blocks that are executed cyclically until an exit condition becomes true (what would be a loop in a programming language). There are two kinds of exit conditions: ranges, which establish a variable and a set of values to go through, and stop-when conditions which are boolean expressions. When a variable reaches the last value of its range or a stop-when condition is satisfied, the iteration ends.
- **Alternation:** an alternation is a sequence of execution blocks which are executed if a set of conditions is satisfied (what would be an *If* in

a programming language). These conditions are called guards, and the alternation is executed if all of them are satisfied. Guards are boolean expressions, and they are described in the next section.

3 User Guide

3.1 Experiment execution

The experiment execution in PIPE2 follows the subsequent steps:

1. If more than one net is open, the tab corresponding to the one over which the experiment will be run has to be selected. In Figure 1, the selected net is the one called “*Producer and Consumer.xml*”.
2. Select the option ***Load experiment*** from the ***Experiment*** menu on the main window menu bar as shown in Figure 2.
3. Next, an open-file window will appear. Choose the experiment file that has to be executed (see 3).
4. After that, a file in which to save the results needs to be chosen, as shown in Figure 4.
5. Then, the experiment execution will start unless there is a syntactical or semantical error. If that happens, an error dialog box will appear with the corresponding message, otherwise the execution will start and a new dialog with an *Abort* button will appear¹
6. Finally, when the execution finishes, the *Abort* dialog will fade and a new one will appear informing about the location of the file where the results have been stored (the one selected in step 4).

3.2 Experiment Editor

To open the Experiment Editor select the proper option in the *Experiment* menu. An empty experiment window will open. This is the Experiment Main Window (EMW from now on) where first-level elements such as *SolutionSpecs* and *Variables* will be placed using the proper option from the *New* menu.

¹Notice that the execution cannot be aborted while the net is being solved. Once the *Abort* button has been pressed (see Figure 5), the execution of the experiment will be interrupted only when it reaches a check point, such as the beginning of an iteration or the beginning of a *SolutionSpec*.

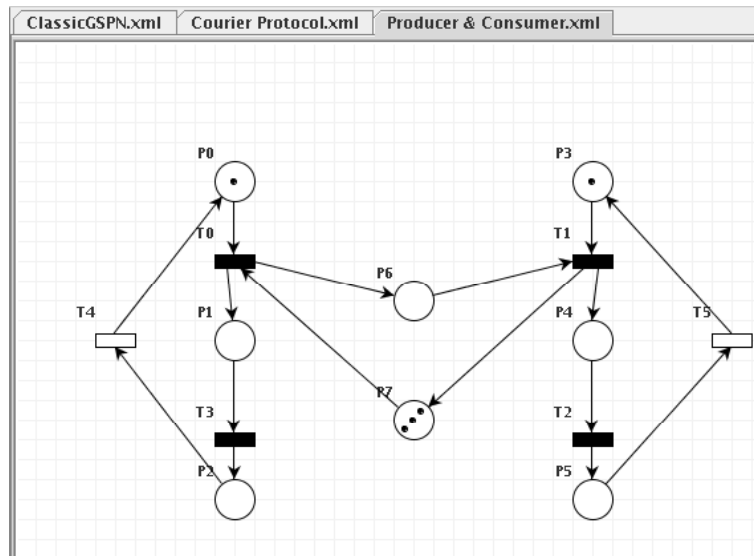


Figure 1: Step 1. Select net.

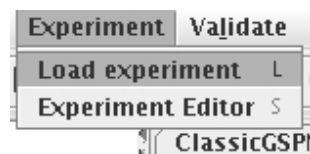


Figure 2: Step 2. Select *Load* option.

An existing experiment can also be opened from the *Open* option in the *File* menu, and modified. The experiment is automatically linked to the selected net, i.e. the net on the selected tab in case that more than one net is open.

3.2.1 Variables

By clicking *Variable* in the *New* menu a variable is added to the experiment and shown as a new panel on the EMW. The first field to fill in is the variable name, the second one is the variable type (*Local* by default), and the rest of the fields depend on the variable type selected. See how these fields change when the variable type is changed. Next, the different fields that appear in the variable panel depending on the variable type chosen are described:

- **Local** (see Figure 6)
 - Initial value: it represents the *InitialValue* attribute of the *Local-Variable* element from Ex-SE format. The local variable will take this value at the beginning of the experiment execution.

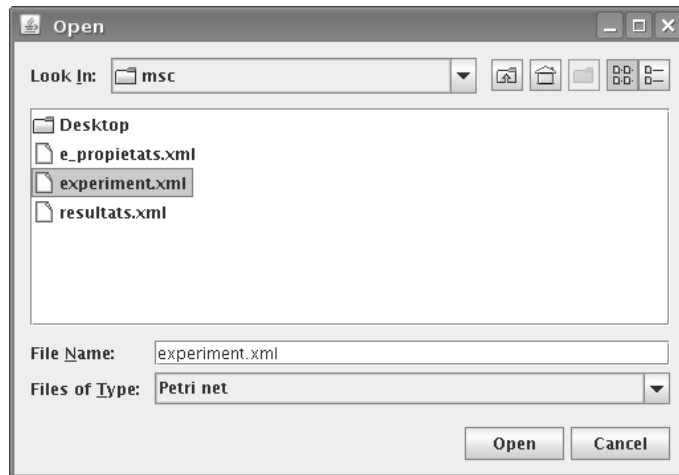


Figure 3: Step 3. Select experiment file.

- **Global** (see Figure 7)
 - Element type: type of the referred element. It can be *Place*, *Transition* or *Arc*.
 - Element: combobox with all the selectable elements. The content of this combobox depends on the element type selected. For instance, if *Transition* type has been chosen, the combobox will contain all the transitions belonging to the selected Petri net.
 - Attribute: it also depends on the element type selected. For example, if *Transition* type has been selected, the *Attribute* combobox will contain transition-characteristic attributes. The value of the *Attribute* is linked to the variable in such a way that when the value of this variables is changed, the value of this attribute will also change.
- **Output** (see Figure 8)
 - Element type: type of the element which this variable will refer to. It can be *Place* or *Arc*.
 - Element: combobox with all the selectable elements. The content of this combobox depends on the element type selected. For instance, if *Transition* type has been chosen, the combobox will contain all the transitions belonging to the selected Petri net.
 - Result: it depends on the element type chosen. If is is a *Transition*, the combobox will contain all the transition-characteristic

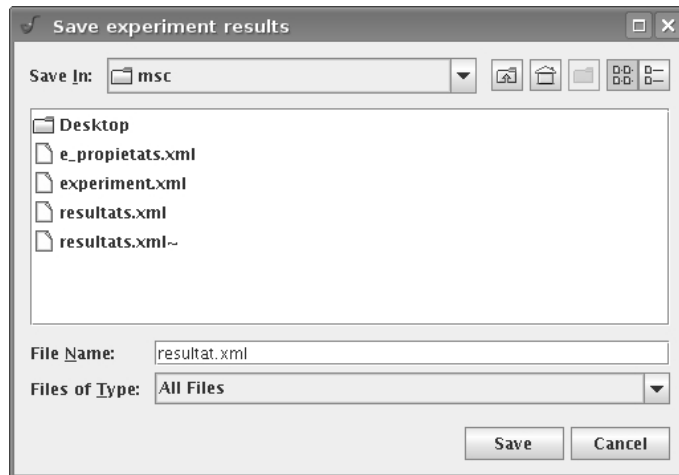


Figure 4: Step 4. Choose results file.



Figure 5: Abort dialog.

outputs, for example throughput. If the element type is *Place* the combobox will contain all the place-characteristic output, such as utilization or average number of tokens. The value of the output is updated after every execution of a *Solve* element.

- Initial value: value of the output variable at the beginning of the experiment execution.

The *Delete* button removes the variable from the experiment, that is, the variable panel from the EMW.

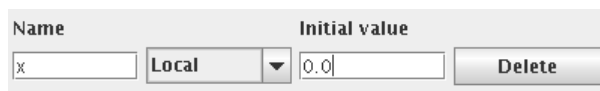


Figure 6: Local variable.



Figure 7: Global variable.



Figure 8: Output Variable.

3.3 SolutionSpec

When clicking *SolutionSpec* in the *New* menu, a *SolutionSpec* is added to the experiment and shown as a new panel on the EMW.

A *SolutionSpec* contains a sequence of execution blocks. To create or edit this sequence, click **Edit** button, and a new window will open (see Figure 9). This is the *SolutionSpec* window (SSW from now on) in which execution blocks (iterations, alternations, solve elements, and assign elements) can be added using the *New* menu.

The **Edit OutputSpec** button opens another window that contains the desired outputs that will be written in the results file for the current *SolutionSpec* scope, (see Figure 10).

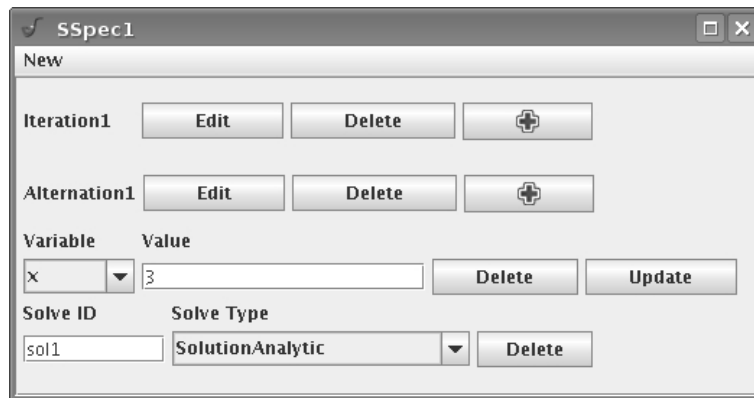


Figure 9: SolutionSpec.

3.4 Alt

An alternation can be placed inside a *SolutionSpec*, inside an iteration, or inside another alternation. The alternation is represented as a panel in the

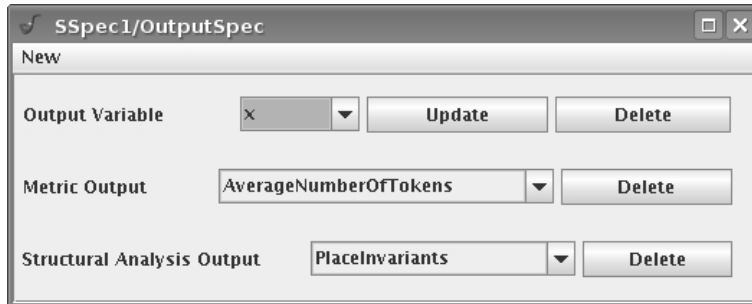


Figure 10: OutputSpec.

parent window, which can be a *SolutionSpec* window, an *Iteration* window, or an *Alternation* window. This panel contains three buttons: the first one is to edit the alternation, the second one is to delete it, and the last one is to view its content. The **Edit** button opens a window (*Alternation* window) where alternation components can be added to or removed from. These components are the guards and the execution blocks. Guards are boolean expression and execution blocks can be assign elements, solve elements, iterations and alternations. These execution blocks will execute only if all the guards are satisfied.

To add a guard to an Alternation one needs to click the **Guard** option from the **New** menu in the *Alternation* window. The execution blocks are added in the same way.

3.4.1 Guard

As part of an alternation, a guard is represented as a panel inside an *Alternation* window with a text field where a boolean expression can be entered (see Figure 11). This expression needs to follow the grammar defined as part as the Ex-SE format (see the grammar definition at A).

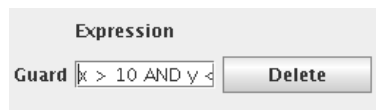


Figure 11: Guard.

3.5 Iteration

An iteration can be placed inside a *SolutionSpec*, inside an alternation, or inside another iteration. The iteration is represented as a panel in the parent

window which can be a *SolutionSpec* window, an *Iteration* window, or an *Alternation* window. This panel contains 3 buttons: the first one is to edit the iteration, the second one is to delete it, and the last one is to view its content. The **Edit** button opens a window (*Iteration* window) where iteration components can be added to or removed from. These components are ranges, *stop-when* conditions and execution blocks. Ranges define a set of values which a variable will go through. *Stop-when* conditions are boolean expressions matching the Ex-SE grammar at A. Execution blocks can be assign elements, solve elements, iterations and alternations. These execution blocks will keep executing in the same order as they are in the experiment file until a variable reaches the end of its range or a *stop-when* condition is satisfied.

To add a range to an iteration, choose **Range** option or **VectorRange** option from the **New** menu of the Iteration window. To add a stop-when condition, choose the **StopWhen** option from the same menu. The execution blocks are added the same way.

3.5.1 Range

An iteration can have two different kinds of ranges: ranges that go through a set of numbers from the first one to the last one increasing (or decreasing) by a certain step at each loop of the iteration (*Range*), and ranges which go through a sorted vector taking its values one by one (*VectorRange*).

A *Range* consists of 5 fields (see Figure 12):

- **Exit type**: defines the step type. It can be **Step** or **StepPercent**. If **Step** is chosen, the value of the **Step** field will be added to the variable at the end of each loop of the iteration. If **StepPercent** is chosen, the variable will increase by the percentage corresponding to the value of the **Step** field.
- **Variable**: name of the variable which will go through this range. It is a combobox containing all the experiment variable names. This list is updated by clicking the **Update** button in case that a variable has been added to the experiment after creating this range element.
- **Start Value**: initial value of the range. It is a real number.
- **End Value**: last value of the range. When the value of the variable is greater than (less than in case of a decreasing range) or equal to this value, the range is considered completed. It is a real number.

- **Step**: step by which the variable is increased at each loop. Depending on the value of the **Exit type** field, this increase will be arithmetic or a percentage.

| | Exit type | Variable | Start value | End Value | Step | | |
|------|-----------|----------|-------------|-----------|------|--------|--------|
| Exit | Step | N | 1 | 2 | 1 | Delete | Update |

Figure 12: Range.

A *VectorRange* consists of 2 fields (see Figure 13):

- **Variable**: it sets the variable which will go through the vector. It is a combobox containing all the experiment variable names. This list is updated by clicking the **Update** button in case that a variable has been added to the experiment after creating this range element.
- **Vector**: vector which the variable will go through . It must contain a list of real numbers separated by commas.

| | Variable | | |
|--------|----------|-----------------------|---------------|
| Vector | q1 | 1.0, 5.0, 3.1, 0.5, 3 | Update Delete |

Figure 13: VectorRange.

3.6 Assign

An assign element can be placed in a *SolutionSpec*, in an alternation or in an iteration (see Figure 14). An assign element consists of the following fields:

- **Variable**: it sets the variable which will take the value of the right part of the assignment. It is a combobox containing all the experiment variable names. This list is updated by clicking the **Update** button in case that a variable has been added to the experiment after creating this assign element.
- **Value**: mathematical expression which result will be assigned to the variable. This expression has to match the mathematical expression grammar defined as part of the Ex-SE format (see A).

| Variable | Value | | |
|----------|---|--------|--------|
| clients0 | <input type="text" value="clients0+1"/> | Delete | Update |

Figure 14: Assign.

3.7 Solve

A *Solve* element can be placed in a *SolutionSpec*, in an alternation or in an iteration (see Figure 15). A *Solve* element consists of the following fields:

- ***Solve ID***: solution identifier. It will allow the user (or a results interpreter) to match the results from the results file with the corresponding *Solve* in the experiment file.
- ***Solve type***: the type of analysis the experimenter will carry out using the PIPE tool. The possible values are (see [2, 1] for detailed description of the different types of analysis):
 - ***SolutionAnalytic***: the model will be solved analytically using GSPN analysis, to obtain performance output metrics, as for example transition throughput.
 - ***PlaceInvariantAnalysis***: place invariants will be computed.
 - ***TransitionInvariantAnalysis***: transition invariants will be computed.
 - ***MinimalSiphons***: minimal syphons will be computed.
 - ***MinimalTraps***: minimal traps will be computed.
 - ***StructuralPropertyCheck***: structural properties will be computed, including boundedness, deadlock-freedom and safety.

| Solve ID | Solve Type | |
|----------------------|------------------|--------|
| <input type="text"/> | SolutionAnalytic | Delete |

Figure 15: Solve.

3.8 eDSPN Nets Import/Export

To import a net from a eDSPN file, the *Import* option from the *File* menu in the PIPE main window has to be chosen.

To export a net to a eDSPN file, choose the *Export* option from the *File* menu in the PIPE main window, and then choose eDSPN format as shown in Figure 16.

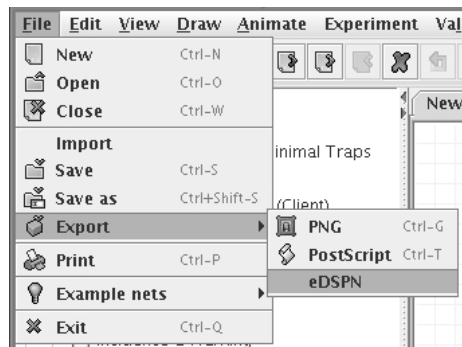


Figure 16: Export.

A Expressions grammar

```
exp                -> math_exp | bool_exp

math_exp           -> NUMBER
                  | math_exp + math_exp
                  | math_exp - math_exp
                  | math_exp * math_exp
                  | math_exp / math_exp
                  | math_exp ** math_exp
                  | (math_exp)
                  | -math_exp
                  | VARIABLE

bool_exp           -> math_exp < math_exp
                  | math_exp <= math_exp
                  | math_exp = math_exp
                  | math_exp >= math_exp
                  | math_exp > math_exp
                  | math_exp <> math_exp
                  | bool_exp AND bool_exp
                  | bool_exp OR bool_exp
                  | (bool_exp)
                  | NOT bool_exp
                  | TRUE
                  | FALSE
```

NUMBER represents the set of real numbers. The integral part and the fractional part are separated by a decimal point. Besides, exponential expression such as 3.0E3 (which is equivalent to $3 \cdot 10^3$) are allowed in this grammar. The following regular expression contains all these numbers:

$$([0-9]+ | [0-9]+ "." [0-9]* | "." [0-9]+) ("E" [0-9]+)?$$

VARIABLE represents the set of values that a variable name can take in this grammar. A well-formed variable name is a sequence of alphanumerical characters (from 'a' to 'z' including capital letters and '0' to '9') as long as the first character is alphabetical:

$$[A-Za-z][A-Za-z0-9]^*$$

References

- [1] Platform Independent Petri net Editor 2. <http://pipe2.sourceforge.net/>.
- [2] P. Bonet, C. Lladó, R. Puigjaner, and W. Knottenbelt. PIPE v2.5: A petri net tool for performance modelling. In *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, October 2007.
- [3] M. Melià, C. Lladó, C. Smith, and R. Puigjaner. Experimentation and output interchange for petri net models. In *Seventh International Workshop on Software and Performance, WOSP08*, June 2008.
- [4] C. Smith, C. Lladó, R. Puigjaner, and L. Williams. Interchange formats for performance models: Experimentation and output. In *Proc. of the Fourth International Conference on the Quantitative Evaluation of Systems*, pages 91–100, September 2007.