The PIPE2 Experimenter

Marc Melià[1], Catalina M. Lladó[1], Connie U. Smith[2], and Ramon Puigjaner[1]

[1] Universitat de les Illes Balears, Departament de Ciències Matemàtiques i Informàtica, Ctra de Valldemossa, Km. 7.6, 07071, Palma de Mallorca, Spain
marcnma@gmail.com, cllado@uib.es, putxi@uib.es
[2] Performance Engineering Services, PO Box 2640 Santa Fe, New Mexico 87504-2640, USA
www.spe-ed.com

Abstract. Performance model interchange formats are common representations for data that can be used to move models among modeling tools. The Experiment Schema Extension (Ex-SE) is a format that provides a means of specifying performance studies (model runs and output). It is independent of a given tool paradigm, e.g., it works with queuing networks, Petri nets and other paradigms. PN-Ex is an extended instantiation of the Ex-SE for Petri nets. This paper presents an Ex-SE based Experimenter for PIPE2, a Petri net tool that provides structural and quantitative analysis modules. Two case studies demonstrate the use of the format and the Experimenter.

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.

For example, interchange formats have been specified for: queueing network models: PMIF (*Performance Model Interchange Format*) [7], LQN (*Layered Queueing Networks*) [3], UML (*Unified Modelling Language*) [9], Petri Nets, and others. 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 [8] 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. In [8], the use of the Ex-SE is illustrated with an instance of the extension in which the interchange schema is the PMIF, called PMIF-Ex.

Petri nets are different from the other paradigms, however, because they provide additional representation and analysis capabilities in addition to performance analysis. Examples include constraints on tokens in places, invariant

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 980-989, 2008.

analysis, reachability analysis, and so on. A specific, extended instantiation of the Ex-SE for Petri nets (PN-Ex) is presented in [6]. This paper demonstrates the viability of the approach with an Experimenter for PIPE2 (Platform Independent Petri net Editor 2) [4, 1], a Petri net tool that provides structural and quantitative analysis modules. A case study shows the use of the format and the Experimenter.

The paper first presents an overview of the Experiment Schema Extension (EX-SE) followed by a description of the specific extensions in the Petri Net instance (PN-Ex). Next, the Experimenter that was developed to prove the concept is described, and two case studies illustrate the usefulness of the approach.

2 Experiment Schema Extension (Ex-SE)

The Experiment Schema Extension (Ex-SE) is an extension to performance model interchange formats for defining a set of model runs and the output desired from them [8]. It incorporates elements from each of these approaches to provide a comprehensive solution for experimentation. The Ex-SE provides a means of specifying performance studies that is independent of a given tool paradigm, e.g., it works with PMIF, S-PMIF, LQN, and other paradigms. It requires only that a tool support the Ex-SE or have an interface that is capable of reading/writing extended interchange files.

The Experiment schema specifies (A) Variable declaration(s):

- Variables - assign values to an attribute of the model, iterate over it, and so on.
- LocalVariables - expressions that combine variables in solution specifications
- OuputVariables - concrete results from a solution that determine subsequent actions,

(B) Solution specification(s) - the model runs and parameter values desired:

- Assignment - assigns values to model attributes
- Iteration - one or more Range of values to be assigned, StopWhen conditions for termination, and Solve specifications
- Alternation - a test and actions to be taken when true
- Solve - the type of solution desired and the point(s) in the experiment where it should be executed
- ToolCommand - control parameters that are not included explicitly because they depend on the tool,

and (C) Output specification(s) - the variables to be written, the results (e.g., throughput or utilization), and possibly tool specific output.

The experiment schema definition is included in the host schema, e.g., a schema specifying a Petri net model. An OutputFormat schema is also needed in the host schema to specify the XML format to be used for output from the experiments. Thus, the Ex-SE allows specification of:

- Changes in parameter values from one execution of a model to the next
- Specification of control in performing model studies, including iteration and alternation
- Variables that are local to the experiment to be used in computations and output
- Model-results dependent execution
- Use of previous output as input to subsequent runs
- Specification of the output metrics to be returned
- Solution type specifications.

The schema specifies the syntactic characteristics of the Experiment. Additional semantic constraints and assumptions used in Ex-SE are provided at: `www.spe-ed.com/pmif/pmif-ex_readme.htm`.

Petri nets are different from the other paradigms because they provide additional representation and analysis capabilities in addition to performance analysis. Therefore, the specific, extended instantiation of the Ex-SE for Petri nets is described next.

2.1 Petri Net Experiment Specification (PN-Ex)

In PN-Ex, *Variables* can refer to places, transitions or arcs. Places may have capacity limitations or an initial marking, whereas transitions have weights or delays, and priorities. Petri net arcs may have a weight attribute which indicates the number of tokens added to or removed from a place. The results that can be assigned to *OutputVariables* are: throughput for transitions, utilization and average number of tokens for places.

The PN-Ex uses most of the same solution specifications in the Ex-SE, such as assignment, iteration and alternation. However, in addition to the analytic and simulation solution types for the Solve command, the following solution types can also be requested in PN-Ex: PlaceInvariantAnalysis, TransitionInvariantAnalysis, MinimalSiphons, MinimalTraps and StructuralPropertiesCheck.

In the PN-Ex Output specifications the possible values for performance metrics are: TokenProbabilityDensity, Throughput, AverageNumberOfTokens, Utilization, Delay and TimeUnits. On the other hand, the possible values for StructuralAnalysis results are: PlaceInvariants, TransitionInvariants, MinimalSiphons, MinimalTraps, and StructuralProperties.

3 The PIPEv2.5 Experimenter

In this section, PIPEv2.5 main features are introduced, followed by the design and implementation of the experimenter and related functions.

3.1 PIPEv2.5

In order to test and improve the experiment format, we selected a Petri Net tool that supports structural and quantitative analysis. Among all tools considered,

PIPE [4, 1] was the one which best fit our needs. It is free, platform independent, and it is continuously improved by several development teams. Besides, it can perform a large number of calculations listed below:

1. GSPN analysis: calculates the average number of tokens in a place, the token probability density and the throughput of timed transitions
2. Invariant analysis: computes the place invariant and transition invariant vectors
3. Incidence & marking: displays the forward, backward and combined incidence matrices and the marking matrix and the set of enabled transitions.
4. Reachability graph: provides a visual representation of all the possible firing sequences for the given Petri net. It also gives information about boundedness, safeness and deadlock-free properties.
5. Simulation: computes the average number of tokens per place along with the 95% confidence interval for each place in the net.
6. State space analysis: builds a tree of all the reachable markings which is used to determine boundedness, deadlock-free, and safeness. It also provides the shortest path to deadlock if there is one.
7. Minimal Siphons and minimal traps: computes sets of places called siphons and traps for structural analysis.
8. Others: classification, model comparison, DNAmaca interface

3.2 Experimental Framework Design

The PIPE tool has been adapted by adding three new features described in the following paragraphs: experiment editing, experiment validation and execution, and PN format import and export. More detailed description of how to use the experimental framework can be found at `dmi.uib.es/~cllado/ex-se/`.

Experiment editing The tool incorporates an experiment editor GUI which allows the user to create and edit experiments. The experiment editor automatically provides the user with place names, transition names, and arc names that are part of the net, as well as previously declared variable names.

The GUI, shown in Fig. 1 consists of a main dialog which handles the creation of new solution specifications and variable declarations. Each solution specification consists of two dialogs, one for the output specification and the other for simple execution blocks, alternation blocks, and iteration blocks containing ranges and *stop-when* conditions.

Experiment validation and execution An experiment is performed in 2 steps:

1. Validation. The experiment document is loaded and syntactically validated against the schema.
2. Execution. While parsing the XML document (using a DOM parser), the experiment is executed as follows:

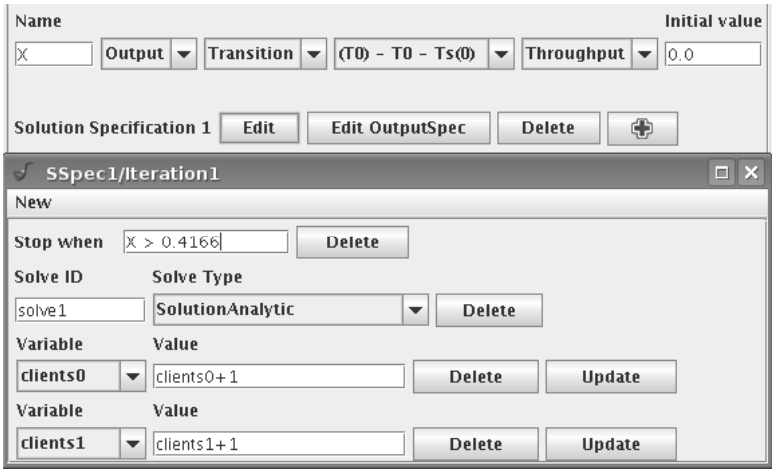

Fig. 1. Experiment edition with PIPEv2.5

(a) *Variables*, *OutputVariables* and *LocalVariables* are created and initialized.
(b) For each solution specification, *Variables* and *Output Variables* are reset. For valid execution blocks:
   i. *Assignment*: the expression is calculated and assigned to the variable. If the target variable has an *attribute to change*, the net is updated with the new value.
   ii. *Alternation*: appropriate alternation sub-blocks are executed
   iii. *Iteration*: all variables referenced in range elements are initialized to the *start-value*. The iteration sub-blocks are executed repeatedly until a *stop-when* condition is satisfied or any of the range-variables reaches its *end-value*. After each cycle, all range-variables are updated according to their *step-value*.
   iv. *Solve*: The appropriate solution is executed, the results are written to the output file according to the output specification, and the output variables are updated.

Import and Export PIPE native format was not the most suitable one for experiments. Therefore, the well suited TimeNet (TIMEd Net Evaluation Tool) [2] e-DSPN (extended Deterministic and Stochastic Petri Nets) format is used with PN-Ex for the experiment specification. Hence, we gave PIPE the capability of importing and exporting this format. We also extended the e-DSPN format for PIPE to include arc weights and place capacities which were not considered in eDSPN.

Since PIPE's internal representation of both Petri net formats is the same, the same experiment file can be used with both PIPE-native and eDSPN files.

4 Case Studies

Two case studies show the use of the format and validate the Experimenter. The first one shows the step by step creation of an experiment with the PIPE Experimenter showing the results obtained.

The second one takes a published model and experiment description and compares its results with results obtained automatically with the Experimenter.

4.1 A Multiserver Cyclic Polling System

We use the GSPN description of a single-server cyclic polling system published in [5] to show a step by step creation and execution of an experiment in PIPE. The system consists of a set of waiting lines that receive arrivals from an external source, and one server that cyclically visits the queues, providing service to any waiting customers. Our GSPN model, shown in Fig. 2, represents two waiting lines, for simplicity. Table 1 shows the characteristics of places and transitions of the model.

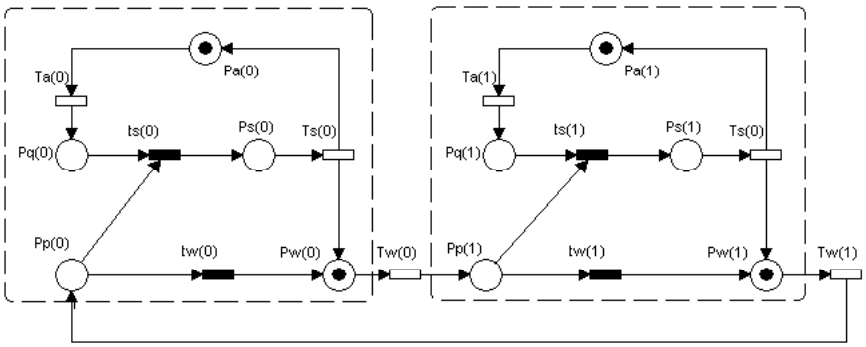

Fig. 2. GSPN representation of a single/server cyclic polling system

timed transitions	weight	immediate transitions	weight	priority
Ta(0),Ta(1)	$1.0s^{-1}$	ts(0),ts(1)	1	2
Ts(0),Ts(1)	$1.0s^{-1}$	tw(0),tw(1)	1	1
Tw(0),Tw(1)	$5.0s^{-1}$			

Table 1. Characteristics of the transitions in the GSPN model of a cyclic single-server polling system

Transitions $Ta(0)$ and $Ta(1)$ model the customer arrival process at queues 0 and 1 respectively. Transitions $Tw(1)$ and $Tw(2)$ model the server walk time

from queue 1 to queue 2 and viceversa. Customers waiting for a server are queued in place $Pq(0)$ and $Pq(1)$, while a token in place $Pp(0)$ and $Pp(1)$ represents the server polling queue 0 and 1. A token in places $Ps(0)$ and $Ps(1)$ represents serving a customer at queue 0 and 1 respectively.. The number of customers in each queue is the initial marking in place $Pa(0)$ and $Pa(1)$ for queues 0 and 1. In our experiment, we will increase this number at both queues in parallel until a value very close to the maximum throughput is reached. When the system is fully loaded (a customer is always waiting for service) the server will provide service at each queue, which implies a mean cycle time equal to 2.4 s (2.0 s for service and $((1/5) * 2 = 0.4$ s to walk), then the maximum throughput that can be observed is $1/2.4 = 0,41\hat{6}\ s^{-1}$ for each queue (throughput of $Ta(0)$ and $Ta(1)$). Therefore, our experiment will start with one customer at each queue, and increase the number of customers until the throughput reaches this value. We are happy with 4 digits of accuracy, so, the experiment will run until the throughput passes the value $0,4166$.

In PIPE we first create the GSPN model Next the variables are created as shown in the top part of Fig. 1. In this case, we need two *Variables* to increment the number of customers for queues 0 and 1 (called *clients0* and *clients1*) and an *Output Variable* (shown in the figure) to store the throughput value obtained in the previous execution so it can be compared with the maximum value. Next we create the variables for the performance results to be printed in the output file. Fig. 3 shows the experiment editor once those are created.

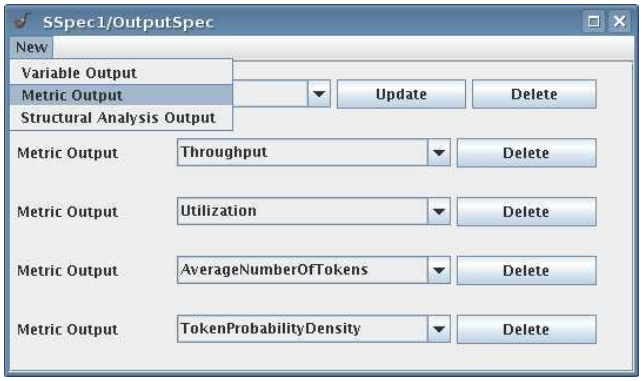

Fig. 3. Output specification for the experiment of a cyclic single-server polling system

Then, we create the solution specification which in this case contains one *Iteration* element with a *StopWhen* condition and two assignments to increment the number of customers at each queue. Since we do not want to limit these values a *Range* cannot be used. The following shows the xml specification that is generated using the experiment editor:

```
<SolutionSpec>
    <OutputSpec>
```

```
        <WriteVariable VariableName=''clients0''/>
        <WriteVariable VariableName=''clients1''/>
        <WriteVariable VariableName=''X''/>
        <WriteOutput Metric=''Throughput''/>
        <WriteOutput Metric=''Utilization''/>
    </OutputSpec>
    <Iteration>
        <StopWhen Test=''X > 0.4166''/>
        <Solve SolutionID=''solve1''>
            <SolutionAnalytic/>
        </Solve>
        <Assign VariableName=''clients0'' Value=''clients0+1''/>
        <Assign VariableName=''clients1'' Value=''clients1+1''/>
    </Iteration>
</SolutionSpec>
```

Finally, the experiment is loaded and run. An excerpt of the results of the last run is as follows. The throughput maximum has been reached and a customer is always waiting for service at the queues (utilization of Pq(0) very close to 1).

```
<Solution ID=''solve1''>
   <ValueUsed VariableName=''clients0'' VariableValue=''6.0''/>
   <ValueUsed VariableName=''clients1'' VariableValue=''6.0''/>
   <ValueUsed VariableName=''X'' VariableValue=''0.416643''/>
   ...
   <OutputTransition TransitionID=''Ta(0)'' Throughput=''0.416643''/>
   <OutputPlace PlaceID=''Pq(0)'' Utilization=''0.999491''/>
   ...
</Solution>}
```

4.2 A Parallel Communications Software System

In order to validate the approach and Experimenter, we selected a published experiment that provides sufficient data on the model, the output, and the experiment for reproducibility. The experiment is published in Woodside and Li 91 [10] and represents the passing of messages following the *Courier* protocol. Fig. 4 shows the software tasks and the data flows between them. A message originates at the Sender Side User task, and is conveyed, following the arrows, to the Receiver Side User task. The message is processed by two tasks implementing the ISO Session and Transport layers at each end, and is conveyed between users and layer tasks by *Courier* tasks.

The detailed GSPN model for one-way data transfer and its characteristics is given in [10]. The published experiment varies the transport window size, called N and the fragmentation ratio of data at the transport level, called $(q1 : q2)$. The result presented is the throughput rate λ from the users' point of view. Table 4 compares results published in [10] with results obtained with PIPE in one experiment, and for comparison, the ones obtained with TimeNET. TimeNET results for the 3^{rd} run are not shown because it ran for more than 24 hours on three occasions and never ended.

The experiment in PIPE (1 single execution with 4 runs of the model) on a Pentium IV at 3GHz and with 512MB of DRAM ran for 20 minutes and 30 seconds. The run time for TimeNET on the same machine (4 different runs of the tool changing the corresponding parameters) is substantially longer as shown in Table 4.

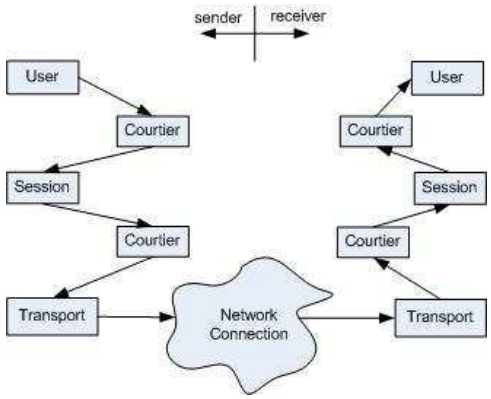

Fig. 4. Tasking model of courier protocol software

It can be seen from the table that output measures have the same value for the first three runs but that surprisingly, for the forth run (N= 2, q_1:q_2=2:1) the output values published in [10] are noticeably different. This is the reason why we decided to also run the experiment with TimeNET. Results show that there is probably some sort of error in the published work since PIPE and TimeNET give basically the same result. It is not the first time that running experiments in an experimental framework discovers wrong results in published material (see [8]). This shows that it is easy to introduce errors in the models when different executions for each combination of model parameters have to be done. It is less likely when the experiment (different runs of the model) can be automated. This is an additional advantage of an experimental framework.

Experiment	**by**	λ	Time
$N = 1$ $q_1 : q_2 = 1 : 1$	Woodside and Li 91	74.346664	
	PIPE	74.346677	
	TimeNET	74.346594	108 seconds
$N = 1$ $q_1 : q_2 = 2 : 1$	Woodside and Li 91	50.432453	
	PIPE	50.432461	
	TimeNET	50.432406	114 seconds
$N = 2$ $q_1 : q_2 = 1 : 1$	Woodside and Li 91	120.372086	
	PIPE	120.372102	
	TimeNET		
$N = 2$ $q_1 : q_2 = 2 : 1$	Woodside and Li 91	92.875839	
	PIPE	81.052649	
	TimeNET	81.041052	15 hours

Table 2. Validation results of Woodside and Li 91 experiment

5 Conclusions

The Experiment Schema Extension (Ex-SE) is an XML interchange schema extension that defines a set of model runs and the output desired from them. It provides a means of specifying performance studies that is independent of a given tool paradigm. PN-Ex is a specific, extended instantiation of the Ex-SE for Petri nets. In this paper, we have demonstrated the viability of the approach with an Experimenter for PIPE2, a Petri net tool that provides structural and quantitative analysis modules. Two case studies have shown the use of the format and the Experimenter.

It would be relatively easy to adapt the Experiment Editor to use it with other instances of the EX-SE, since some parts of it always stay the same. For example, an adaptation to use it with the PMIF-Ex [8] would only require a modification of the Variable editor (just changing the type of variables) and the Output Specification editor. The Solution Specification editor would be almost the same. In the near future, we would like to address the creation of a general purpose Experimenter tool.

References

1. Platform Independent Petri net Editor 2. `http://pipe2.sourceforge.net/`.
2. TimeNET 4.0. TIMEd Net Evaluation Tool. `http://pdv.cs.tu-berlin.de/~timenet/`.
3. `http://www.sce.carleton.ca/rads/lqn/lqn-documentation/`.
4. P. Bonet, C. Lladó, R. Puijaner, 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.
5. M. Marsan, A. Bobbio, and S. Donatelli. *Lectures on Petri Nets I: Basic Models*, chapter Petri Nets in Performance Analysis: An Introduction, pages 211–256. Springer, 1998.
6. 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.
7. C. Smith and C. Lladó. Performance model interchange format (PMIF 2.0): XML definition and implementation. In *Proc. of the First International Conference on the Quantitative Evaluation of Systems*, pages 38–47, September 2004.
8. 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.
9. C. U. Smith, V. Cortellessa, A. Di Marco, C. M. Lladó, and L. G. Williams. From uml models to software performance results: An spe process based on xml interchange formats. In *Proc. of the Fith Workshop on Software and Performance (WOSP'05)*, pages 87–98, July 2005.
10. C. Woodside and Y. Li. Performance petri net analysis of communications protocol software by delay-equivalent aggregation. *Petri Nets and Performance Models, 1991. Proceedings of the Fourth International Workshop on*, pages 64–73, 1991.