Interchange Formats for Performance Models: Experimentation and Ouput*

Connie U. Smith
Performance Engineering Services
PO Box 2640 Santa Fe
New Mexico, 87504-2640 USA
www.spe-ed.com

Catalina M. Lladó, Ramon Puigjaner
Dep. de Cien. Matemàtiques i Informàtica
Universitat de les Illes Balears
07071, Palma de Mallorca, Spain.
cllado@uib.es, putxi@uib.es

Lloyd G. Williams
PerX
2345 Dogwood Circle
Louisville, CO 80027 USA
www.perfx.net

Abstract

XML-based interchange formats for performance models provide a mechanism whereby performance model information may be transferred among modeling tools. For example, the PMIF allows diverse tools to exchange queueing network model information. Formats have also been defined for the interchange of LQN, UML, 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. This paper presents an XML interchange schema extension for defining a set of model runs and the output desired from them. It has the expressive power to specify iterations, alternations, assignments of values, actions based on model results and more. Examples illustrate how the experiment interchange extension can be used with a wide variety of performance modeling paradigms. A prototype proves the concept.

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, the Performance Model Interchange Format (PMIF) [**?**, **?**] allows diverse tools to exchange queueing network model information. Use of the PMIF does not require tools to know about each others capabilities, internal data formats, or even existence. It requires only that the importing and exporting tools either support the PMIF or provide an interface that reads/writes model specifications from/to a PMIF file. Interchange formats have also been defined for layered queueing networks (LQN), UML, Petri Nets and other types of models.

In each interchange format, a file specifies a model and a set of parameters for one run. Since modeling studies typically require multiple runs of the same model using different parameters (e.g., different workload mixes), this requires preparing and exchanging multiple interchange files. In addition, interchange formats do not specify the output metrics that are to be returned after model execution, typically resulting in either a default set of metrics or all possible metrics.

To address these issues, this paper presents an Experiment Schema Extension (Ex-SE) for defining a set of model runs and the output desired from them. This schema extension provides a means of specifying performance studies that is independent of a given tool paradigm. It requires only that a tool support the Ex-SE or have an interface that is capable of reading/writing extended interchange files. This schema extension was developed for use with an interchange schema (e.g., PMIF) when exchanging models between performance modeling tools. However, it may also be used in a stand-alone mode to specify studies for the tool in which the model was created. It may also be used to

specify measurement as well as modeling studies.

To illustrate the use of the Ex-SE, this paper defines and uses it with the PMIF. Thus, this instance of the extension is known as PMIF-Ex.

The contributions of this work are:

- Definition of a modeling-paradigm independent schema for specifying experiments
- A set of examples illustrating how experiments may be specified using Ex-SE
- Implementation of a prototype that demonstrates the feasibility of this approach
- Demonstration that the approach works with multiple modeling paradigms

We begin with a summary of related work. Section 3 presents the schema and provides examples to illustrate how to use it to express experiments. Section 4 discusses a prototype PMIF-Ex implementation. Section 5 describes a case study with an actual, large-scale model, and compares some results originally produced for that project with those derived automatically using the PMIF-Ex. Section 6 demonstrates how the Ex-SE can be used with other modeling paradigms. Summary and conclusions are in section 7.

2. Related work

Related work falls into three categories: model interchange formats, experimentation, and tool interfaces.

2.1. Model interchange formats

Several model interchange formats for different types of models have been defined. The most relevant of these are described below. PMIF 1.0 [**?**] and 2.0 [**?**] both allow various tools to exchange queueing network model information. Both PMIFs are based on meta-models, which provide an underlying formalism for the schemas. PMIF 1.0 uses EDIF/CDIF [**?**] for the transfer format. PMIF 2.0 is based on a revised and expanded meta-model and uses XML for the transfer format.

Other approaches have focused on transferring information between UML-based software design tools and software performance engineering tools. For example, Cortellessa and Mirandola annotate UML diagrams and transform them into Execution Graphs and Queueing Network Models [**?**]. Cortellessa, et. al., have implemented this approach using multiple XML files: one with the workload specifications and another with the device specifications for the Queueing Network Model [**?**]. More recently Cortellessa, di Marco, Lladó, Smith and Williams collaborated on a unified approach using S-PMIF [**?**], derived from the original SPE meta-model developed along with the original PMIF [**?**].

Gu and Petriu use XSLT (eXtensible Stylesheet Language for Transformations) [**?**] to transform UML models in XML format to the corresponding Layered Queueing Network (LQN) description which can be read directly by existing LQN solvers [**?**]. Wu and Woodside use an XML Schema to describe the contents and data types that a Component-Based Modeling language (CBML) document may have [**?**]. CBML is an extended version of the Layered Queuing Network (LQN) language that adds the capability and flexibility to model software components and component based systems. Ambrogio also transfers UML models to LQNs in [**?**]. Balsamo and Marzolla transfer from UML directly to QNM in [**?**]. These approaches again use XML to transfer design specifications into a particular solver; however, they do not attempt to develop a general format for the interchange of queueing network models among different tools.

Woodside et al. developed a meta-model, PUMA, that combines software and system models based on LQN in [**?**].

Harrison et al. [**?**] generalized the PMIF specifications by considering more abstract collections of interacting nodes using concepts compatible with the Reverse Component Agent Theory (RCAT). The interactions are more general in that they synchronize transitions between a pair of nodes rather than describing traffic flows.

The Ex-SE can be easily adapted to work with all of these XML-based approaches. The adaptations are described in Section 6.

2.2. Experimentation

The Ex-SE is based on a meta-model of the information required to conduct a performance modeling study as well as the information produced by such a study. The following work has also addressed the abstract information requirements for modeling studies.

Zeigler [**?**, **?**] has proposed a framework for modeling and simulation that defines the entities and relationships that are central to modeling and simulation. The framework consists of: the *source system*, an *experimental frame*, the *model*, and a *simulator*. In Zeigler's terminology, the source system is the *environment* that is being modeled. An experimental frame is a specification of the conditions under which the system or its model are to be exercised. The specification of an experimental frame includes:

- *Input stimuli*: a description of the class of input stimuli. Actual inputs are drawn from this class.
- *Control*: a description of the conditions under which the model will be initialized, executed and terminated.

- *Metrics*: a description of the output data desired.
- *Analysis*: a description of how the output data will be analyzed to draw conclusions.

A model defines a system for generating input/output behavior and the simulator is any computational system that can execute the model.

Hillston [**?**] describes the IMSE Experimenter, a tool designed to facilitate performance modeling studies within the Integrated Modeling Support Environment (IMSE). The Experimenter uses *experimental frames*, derived from those of Zeigler, to specify a single execution for a given model. Each experimental frame includes input, output, and control specifications. The Experimenter adds the notion of an *experiment* to allow specification of studies involving multiple models with (possibly) multiple executions per model. Experiments are described by *experimental plans* which make it possible to group model executions by objective.

The IMSE Experimenter also allows the user to specify how the values of each parameter in a model's input specification vary during an experiment. Finally, the experimental plan must include at least one *analysis* specification that describes how experimental *results* are obtained from model outputs. Thus, outputs from multiple runs can be used to obtain overall measures (e.g., mean, standard deviation).

The Software Performance Experimenter (SPEX) is a tool for managing performance studies using LQN models [**?**]. SPEX also has provisions for input, output and control specifications. Input parameters may be specified using any legal Perl expression. *Observation indicators* are used to indicate output metrics of interest (e.g. utilization, throughput). These metrics may then be written to the result file. Control statements allow the user to specify stopping conditions (e.g., convergence limit), the solver to be used, and where the experiment is to be run. Array input parameters allow specification of sequence and iteration conditions but not alternation.

We have incorporated elements from each of these approaches into the Experiment Schema Extension to provide a comprehensive solution for experimentation.

2.3. Tool interfaces

Tool interfaces may also include experimentation capabilities that define the input, output, and control information that can be used to perform modeling studies. Table 1 summarizes the capabilities of several tools that provide experimentation capabilities.

The tools selected for the table are representative of the types of products used for *model experimentation*. Most current tools have a Graphical User Interface (GUI) that leaves it to the user to conduct model experiments. Tools developed before GUIs were prevalent, however, provided experimentation features as part of their user interface. We have included features provided by some of those historical tools in Table 1. The tools in the table are: Qnap [**?**], Modline [**?**], CSIM [**?**, **?**], MAP [**?**], LQN [**?**, **?**], HIT and HiSlang [**?**, **?**], and the IMSE Experimenter [**?**]. For each tool the table shows the type of model(s) solved by the tool, the type of model interface, the type of experimental support, whether or not there is support for alternation (Alt), and additional experimental support features provided. All of the tools in the table provide support for assigning variables and repetition, so those experimental features are omitted to save space.

Some of the differences between these tool interfaces and the Experiment Schema Extension include:

- The IMSE Experimenter [**?**] and Modline [**?**] include constraints that eliminate combinations of assignments that are not allowed. Our specification is rich enough to specify only allowable combinations so constraints are not necessary.
- Some experimentation mechanisms call for an implicit solution after a sequence of actions is complete. We opted for an explicit Solve element for clarity of specifications.
- We have not included specific experiment debugging mechanisms or other features that custom programming can provide. Some of them could be handled by specific tool commands (see section 3).
- We have not yet included specifications for the analysis of output to produce results. This is a topic for future work. The other features for experimental support in Table 1 are included in Ex-SE either directly or with a ToolCommand specification.

3. PMIF-Ex

This section describes the Experiment Schema Extension for PMIF. We present the schema and provide some examples. This schema definition is included in the host schema (PMIF). An OutputFormat schema (not shown here) is also needed in the host schema to specify the XML format to be used for output from the experiments.

3.1. Schema

As shown in Fig. 1, the Experiment schema has two well-differentiated parts. The first part is the variable specification. This allows the user to specify different types of variables that can be used anywhere in any of the solution specifications. Variables refer to an attribute of an element in

Table 1. Taxonomy of tools supporting experimental solutions

Tool	Model type	Interface	Experiments	Alt	Additional
QNAP2	QNM, Custom	Language	Language	Yes	Custom[1]
Modline	QNM, Custom	GUI	GUI	No	Constraints, Output filtering, Aggregation[2]
CSIM	Custom	C	Language	Yes	Custom[1]
LQN	QNM	GUI, Language	External: SPEX	No	Iteration limit, Hosts, Feedback results, Report specifications
MAP	QNM	Language	Language	Yes	Exec debugging commands, Statistics tables
HIT Hi-Slang	Software, QNM	Language	Language	No	Aggregation[2]
IMSE Experimenter	Software, QNM	Language, GUI	Experimenter	No	Assign random variables, Constraints, Feedback results, Extracted values

the model (for example the ArrivalRate of a specific OpenWorkload). They can be used to assign different values to that attribute, iterate over it, and so on. Expressions combining variables can be assigned to LocalVariables. Finally, an OuputVariable specifies a concrete result that will be used in the solution specification (for example, OutputVariable UCPU represents the Utilization of the node named CPU).

The second part is the solution specification, which indicates the experiment and the output desired. That is, the variables to be written in the output, the results (e.g., throughput or utilization) and possibly tool specific output.

The ToolCommand element allows specification of control parameters that are not included explicitly because they depend on the tool.

The ExperimentType allows for assignments, iterations, alternations and a specific Solve command to specify the point where the model is to be solved. The Solve command also specifies the type of solution (analytical, simulation). Fig. 1 shows in detail the Iteration (but not Alternation due to lack of space). The Iteration requires at least one Range of values and one Solve statement. It can also have one or more StopWhen conditions that may stop the iteration earlier than specified by the Range(s). It can also include assignments, iterations, and alternations.

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.

We initially had a specific definition of the output to be produced. For example, one could specify throughput for a subset of the workloads. This was changed to simplify the OutputSpec; the (future) specification of analysis and results should filter the results if necessary.

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

3.2. PMIF-Ex Example

Fig. 2 illustrates how to express a typical experiments with PMIF-Ex. It shows an excerpt from the specification for the case study presented in section 5. This example illustrates the use of local variables, results testing, setting multiple values in one iteration, the use of multiple StopWhen clauses, and others. The complete PMIF-Ex file for the case study is at `www.spe-ed.com/pmif/realpmif.xml`.

4. Implementation

A prototype interface was implemented to demonstrate the feasibility of the PMIF-Ex concept. The implementation requires a mechanism to create an experiment definition, and a mechanism to interpret the experiment, solve the models accordingly, and return the requested output. In this section, implementation alternatives for each of these are described then the interface implementation for executing experiments using Qnap is presented. In Section 5, the use of the prototype is illustrated with two case studies.

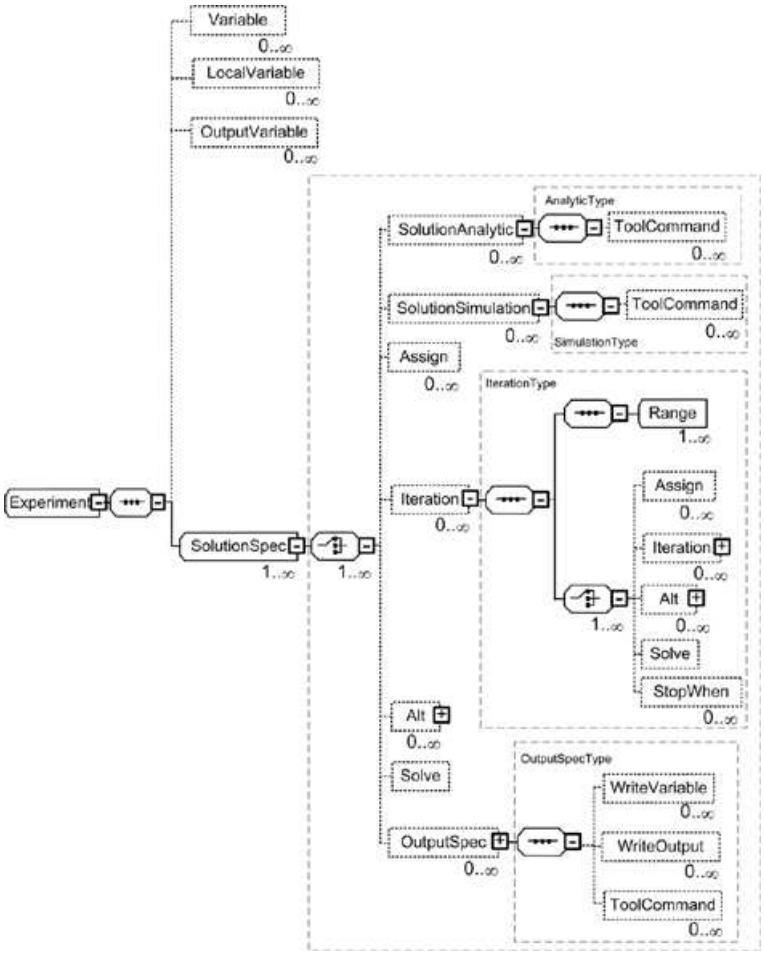

Figure 1. Experiment schema

4.1. Creating the experiment specification

There are three alternatives for creating the experiment specifications:

1. It is relatively easy to use an XML editor, such as XMLSpy, to create experiments for a particular model paradigm.
2. Create a tool with a GUI for describing experiments and generating the XML.
3. Tools that have an experimentation capability could export their experiment definition along with the model interchange format.

We used the first alternative for this proof of concept.

4.2. Executing the experiment

There are two alternatives for interpreting the experiment specifications, solving the models accordingly, and returning the requested output. The choice depends on whether or not the target tool provides its own capability for experimentation.

Tools without experimentation need an Experimenter tool to interpret the experiment, invoke the tool for each `<Solve>`, and return the output. This can be a general tool that can work with multiple solvers.

Tools with experimentation can adapt their Import mechanism to generate the tool specifications for the experiment. This is a more efficient implementation because the tool only needs to be invoked once, and the model does not need to be parsed multiple times. We used this alternative for the proof of concept because Qnap provides direct support for experimentation. Future work will address a general purpose Experimenter tool.

4.3. Prototype implementation

The prototype implementation is based on PMIF and Qnap, a queueing network-based modeling tool. Qnap can

Figure 2. PMIF-Ex example

```
<Variable AttributeToChange="NumberOfJobs" WorkloadName="Forms" Name="NForms" />
<Variable AttributeToChange="NumberOfJobs" WorkloadName="Apply" Name="NApply" />
<Variable AttributeToChange="NumberOfJobs" WorkloadName="Store" Name="NStore" />
<Variable AttributeToChange="NumberOfJobs" WorkloadName="Convert" Name="NConvert" />
<LocalVariable Name="TotTput" InitialValue="0.0" />
<OutputVariable ResultToUse="Throughput" WorkloadName="Forms" Name="TForms" InitialValue=".016" />
<OutputVariable ResultToUse="Throughput" WorkloadName="Apply" Name="TApply"  InitialValue=".005"/>
<OutputVariable ResultToUse="Utilization" ServerName="CPU" Name="UCPU" InitialValue="0" />
<OutputVariable Name="SumTFormTApp" Expression="TForms+TApply" />
<SolutionSpec>
  <Iteration>
    <Range VariableName="NForms" Start="18" End="36" Step="9" />
    <Range VariableName="NApply" Start="32" End="64" Step="16" />
    <Range VariableName="NStore" Start="50" End="100"Step="25" />
    <Range VariableName="NConvert" Start="30" End="60" Step="15" />
    <StopWhen OutputVariableName="UCPU" Test="GE" Value=".99" />
    <StopWhen OutputVariableName="Tform+TApp" Test="LE" Value="TotTput" />
    <Solve SolutionID="Run1">
       <SolutionAnalytic/>
    </Solve>
    <Assign VariableName="TotTput" Value="(TForms+TApply)" />
  </Iteration>
 <OutputSpec>
    <WriteVariable VariableName="TotTput" />
    <WriteOutput Metric="ResponseTime" />
    <WriteOutput Metric="Throughput" />
    <WriteOutput Metric="Utilization" />
 </OutputSpec>
</SolutionSpec>
```

be used on its own or accessed via Modline [**?**]. which provides a graphical, user-friendly interface for model definition and interactive visualization of the results.

The prototype uses Qnap without modification and does not make use of the Modline interface (Qnap2 v.9.3 [**?**]). Qnap reads the input (QNM and experiment specification) from a file and writes the results to another file that also includes the input.

Ultimately, it would be best for Qnap to have an interface that would read from its standard file OR the pmif.xml file. However, we did not have access to the Qnap source code and we could not implement such an interface directly. Therefore, we translated the pmif.xml file into a file in Qnap's format to demonstrate the proof of concept.

The model and experiment translation from a pmif.xml file into a Qnap input file is done using XSLT. We generate a specific XSLT file that transforms a pmif-ex.xml file into a file that can be directly read and executed by Qnap. Additionally, Qnap allows the programming of any input/output operations and so the generation of an XML output file is possible. Some of the detailed issues in the Experiment Schema transformation are as follows:

1. Qnap variables can only be 8 characters, so a routine does this check, truncates the identifier if necessary and generates unique names if it happens to be a duplicate.

2. Iterations are implemented as While statements that finish when any of the StopWhen conditions occur or when any of the Ranges reaches its End value.

3. Alternations are implemented as If statements.

4. Qnap gives a default output that is written to the standard output (typically the screen or a file). The necessary instructions for Qnap to create another file and write the specified results on the pmif-ex.xml in XML format are also generated by the transformation.

5. Proof of concept

Two studies were conducted. The first takes a published model and experiment description and compares its results with results obtained automatically with PMIF-Ex. The second takes an actual case study that was executed by conducting multiple user-driven model experiments using the *SPE·ED* tool [**?**], and compares those results with the automatic experiment results.

5.1. Prototype validation

To validate the prototype, we selected a published experiment that provides sufficient data on the model, the output, and the experiment for reproducibility. The experiment is published in [**?**]. It shows 4 runs of a model presented on page 574-5. The first run is the original model. It is an open model with one CPU and two disks. The second run increases the workload arrival rate. The third explores the advantage of caching by increasing service times, and reducing the demand on one of the disks. The last run uses a lower cost server with only one disk.

Table 2 shows that the results reported in the book match the results derived from the Prototype interchange of the PMIF-Ex model for Runs 1,2, and 4. The results for Run 3, however, differ due to an error in the book. It incorrectly uses 16 CPU visits in calculating CPU demand, rather than the correct value of 12 visits. We only noticed this error after the models produced different results. Even though this is a simple experiment, it demonstrates the successful use of the PMIF-Ex model interchange.

5.2. Case study

To further illustrate the capabilities of the Experiment Schema Extension, we present a second case study that uses an actual complex, large-scale model and compare the results of the original study, performed using *SPE·ED* [**?**] with those of a Qnap execution of a PMIF-Ex specification of the same study.

This study has 4 workloads: Forms, Apply, Store, and Convert. Forms and Apply are user interaction scenarios for submitting some complex information to a Web Server. Upon submission, Store processes the data and archives it in a database. Convert transforms some of the data for storage in an image database. The details of the model are relatively unimportant for this case study because it focuses on taking an experiment performed manually in *SPE·ED*, writes a definition for it, sends it to Qnap, and compares results. i.e., it demonstrates model transformation/exchange and experiment definition/execution. Using a model of an actual, complex, large-scale model, however, does show that the EX-SE can represent the types of studies that are often conducted in practice and shows that they can be conducted faster with EX-SE.

This is a challenging model for solvers because it includes coordination among workloads, large think times (e.g., up to 1 hour), small service demands (e.g., 0.000134 sec.), and long and different execution times (e.g., 20 sec. to 7 min.).

The definition of the experiment is in the bottom portion of Figure 2. The results shown here are a portion of the model runs to determine the scalability of the application. These runs show the baseline platform with 1 CPU and 1 Disk. In the original study these experiments were repeated to study vertical scalability by increasing the number of CPUs and Disks of the Web Server platform. They were then repeated again to increase the number of Web Servers to quantify horizontal scalability. This case study reports a small subset of those extensive experiments.

Table 3 shows the results originally produced with *SPE·ED*, and those produced by the PMIF-Ex translation using Qnap to solve the models. The *SPE·ED* model is an advanced system execution model that explicitly represents coordination among the workloads. It uses a simulation solution. The Qnap model is an approximate model that uses an approximate think time for coordination, and solves the model analytically. Most results are comparable, however, the Store workload shows some significant differences in CPU utilization and response time that require further investigation. The difference is not due to the model transformation; the model parameters are equivalent (other than the think time). Unfortunately, it is not possible to compare to the original measured results because limited data was available. The case study models were created at design time by measuring, modeling, and validating against a prototype.

This example shows the value of having an automated interface for executing the extensive number of experiments. Conducting the original experiments required days of effort because the current *SPE·ED* GUI requires the user to specify and execute each experiment individually. The automated experiment required 18 seconds to run although it would take longer to simulate the models. The proof of concept also shows the value of being able to compare results from different tools using PMIF-Ex. With it the models solved are identical - there are no errors due to manual re-entry of the model into a different tool.

6. Extensions

As noted earlier, the Ex-SE is independent of a given tool paradigm. This paper has described an instance in which the schema was used with the PMIF format. However, to verify its generality, we also investigated the application of the Ex-SE to a number of other model interchange formats.

We found that the Ex-SE is compatible with a large number of interchange formats including:

- S-PMIF - the software performance model interchange format [**?**]
- LQN - the layered queueing network XML definition
- GPMIF - performance model interchange format, compatible with Reverse Component Agent Theory (RCAT) concepts [**?**]
- PNML – Petri Net Markup Language for the interchange of Petri Nets [**?**]
- eDSPN – a Petri Net interchange format, used by TimeNet [**?**] a software tool for the modeling and analysis of stochastic Petri nets with non-exponentially distributed firing times.

In general, one just appends the Experiment type definition into the other XML schema, and changes the Variable type specifications to match new schema.

Table 2. Validation results Jain p.574

		Residence Time			Utilization		
Experiment	**Response**	CPU	DiskA	DiskB	CPU	DiskA	DiskB
Run1							
Jain	1.406	0.0192	0.0345	0.107	0.48	0.42	0.72
Qnap	1.406	0.0192	0.0345	0.107	0.48	0.42	0.72
Run2							
Jain	6.763	0.0278	0.0455	0.750	0.64	0.56	0.96
Qnap	6.763	0.0278	0.0455	0.750	0.64	0.56	0.96
Run3							
Jain	1.013	0.0346	0.0345	0.0546	0.624	0.42	0.40
Qnap	0.754	0.0244	0.0345	0.0546	0.47	0.42	0.40
Run4							
Jain	3.310	0.0192	0.2		0.48	0.90	
Qnap	3.308	0.0192	0.2		0.48	0.90	

Table 3. Validation results for RealStudy

		Response Time		Throughput		CPU Utilization	
Experiment	**Users**	***SPE·ED***	**Qnap**	***SPE·ED***	**Qnap**	***SPE·ED***	**Qnap**
Run1-1							
Forms	18	451	446	0.005	0.005	0.39	0.44
Apply	32	194	177	0.014	0.016	0.15	0.16
Store	50	66	21	0.019	0.017	0.04	0.04
Convert	30	20	21	0.012	0.010	0.06	0.06
Total						0.64	0.70
Run1-2							
Forms	27	1125	1034	0.006	0.006	0.48	0.58
Apply	48	226	243	0.021	0.023	0.21	0.23
Store	75	236	40	0.027	0.026	0.05	0.06
Convert	45	69	61	0.017	0.015	0.09	0.09
Total						0.83	0.96
Run1-3							
Forms	36	2995	3003	0.005	0.006	0.42	0.53
Apply	64	300	457	0.025	0.028	0.24	0.28
Store	100	584	92	0.031	0.034	0.05	0.08
Convert	60	213	184	0.020	0.020	0.11	0.11
Total						0.82	1.00

Since GPMIF is a generalization of PMIF, it can use the Experiment Extension as illustrated for PMIF.

PNML and the eDSPN Schema only need an extended VaribleType specification allowing the reference of places and transitions (instead of nodes and workloads). The extension needed by PNML would be a bit more complicated (or the translation would) since in PNML the elements "places" and "transitions" are specified with only one attribute (ID) and the rest are elements inside those elements. So we cannot just change attributes.

To illustrate the compatibility of Ex-SE with other interchange formats, we illustrate its adaptation to the LQN example in Fig. 3. This example uses the POTS2 case study from [**?**] which was subsequently used as a validation experiment for PMIF in [**?**]. It assumes that POTS2 is formulated as an LQN model (not shown), and shows the definition of an experiment that varies the arrival rate of the four workloads (inside iteration) as the number of CPUs is varied in the outside iteration.

This specification requires changing the LQN schema to use Entryname and Processorname because the LQN schema uses Name for both, and the experiment must differentiate them to know which value to change.

The Experiment Schema Extension definition of Variables must be adapted to the host schema. For example, in PMIF we change the workload intensity by defining a Variable specifying the WorkloadName and the AttributeToChange is ArrivalRate. In LQN, the Variable specifies the Entryname and the AttributeToChange is `arrival_rate`.

The interpretation of the output metrics also changes, but the schema type doesn't have to. For example, with an LQN we would want the utilization by software activities and by devices, not just device.

A GUI experimenter that scans the target schema for the elements and attributes, and lets a user select the variables to be used could handle the variable definitions.

We illustrated the extension to other modeling paradigms

Figure 3. Applying the experiment definition to LQN

```
<Experiment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ExperimentSchema.xsd">
<!-- Note the variable, local variable, and output variable definitions have to change to match LQN schema -->
  <Variable Name="CallO" Entryname="Call_Origination_2" AttributeToChange="arrival_rate"/>
  <Variable Name="CallT" Entryname="Call_Termination_2" AttributeToChange="arrival_rate"/>
  <Variable Name="CalledH" Entryname="Hangup_(Called)" AttributeToChange="arrival_rate"/>
  <Variable Name="CallerH" Entryname="Hangup_(Caller)" AttributeToChange="arrival_rate"/>
  <Variable Name="NumCPU" AttributeToChange="replicas" ProcessorName="CPU" />
  <!-- LQN meta-model uses mean_links for both probability and repetition -->
  <Variable AttributeToChange="mean_links" Name="NumReps" fromActivity="N9" toActivity="N10" />
  <Variable AttributeToChange="mean_links" Name="Node8Prob" fromActivity="N7" toActivity="N8" />
  <Variable AttributeToChange="execution_demand" Name="N1CPU" Activityname="N1" Phasenumber="2" />
  <Variable AttributeToChange="mean_calls" Name="NumCalls" refers_to="3" Activityname="N1" Phasenumber="2"/>
  <OutputVariable Name="CPUUtil" DeviceName="CPU" ResultToUse="Utilization" InitialValue="0"/>
  <SolutionSpec>
     <Iteration>
      <Range VariableName="NumCPU" Start="1" End="4" Step="1"/>
      <Iteration>
        <Range VariableName="CallO" Start="3" End="10" Step="1"/>
        <Range VariableName="CallT" Start="3" End="10" Step="1"/>
        <Range VariableName="CalledH" Start="3" End="10" Step="1"/>
        <Range VariableName="CallerH" Start="3" End="10" Step="1"/>
        <Solve SolutionID="Increase_CPUs,_Increase_Arrival_Rate"/>
           <SolutionSimulation StartInterval="500" StopTime="50000" />
        </Solve>
        <StopWhen OutputVariableName="CPUUtil" Test="GE" Value="1.0" />
      </Iteration>
    </Iteration>
    <OutputSpec>
    <!--These should report the metrics by execution graph node and total for device -->
      <WriteOutput Metric="ResponseTime" />
      <WriteOutput Metric="ResidenceTime" />
      <WriteOutput Metric="Utilization" />
    </OutputSpec>
  </SolutionSpec>
</Experiment>
```

with LQN. The extensions to the interchange formats listed earlier are similar enough that they are not included here.

It is interesting to note that the Experiment Schema Extension is not limited to use with models. It could be used with a measurement experimenter such as DECALS [**?**]. Some of the terminology should be customized, such as changing Solve to Run, and changing the SolutionTypes to something such as ToolSpecification.

7. Summary and Conclusions

This paper has described an Experimental Schema Extension (Ex-SE) for defining performance modeling experiments. The schema allows specification of multiple model runs along with the output that is desired from them. This schema extension provides a means of specifying performance studies that is independent of a given tool paradigm. It requires only that a tool support the Ex-SE or have an interface that is capable of reading/writing extended interchange files. 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 Ex-SE was developed for use with an interchange schema, such as the Performance Model Interchange Format (PMIF), for exchanging models between queueing network based modeling tools. However, it may also be used in a stand-alone mode to specify studies for the tool in which the model was created. It may also be used to specify measurement as well as modeling studies.

This paper has presented a particular instantiation of the Ex-SE, the PMIF-Ex and provided examples of its use. To demonstrate that the concept is feasible, we have also described a prototype implementation of the PMIF-Ex and Qnap. The validity of the prototype was demonstrated with two case studies.

Finally, to verify the generality of this schema, we applied this extension to a number of other model interchange formats (S-PMIF, LQN, GPMIF, PNML, and eDSPN). The

applicability to other interchange formats was illustrated with an example using LQN.

The contributions of this work are:

- Definition of a modeling-paradigm independent schema for specifying experiments
- A set of examples illustrating how experiments may be specified using Ex-SE
- Implementation of a prototype that demonstrates the feasibility of this approach
- Demonstration that the approach works with multiple modeling paradigms

Future work will create the analysis specifications that will transform the Output from the experiment into the desired results, and will address a general purpose Experimenter tool. Future work will also extend the PMIF to include features supported by simulation solvers.

The PMIF-Ex XML schema is available at `www.spe-ed.com/pmif/pmif-exschema.xml`.