Appears in Proc. ACM Workshop on Software and Performance (WOSP) 2010.

# PMIF Extensions: Increasing the Scope of Supported Models

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

Catalina M. Lladó
Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica
Ctra de Valldemossa, Km. 7.6, 07071
Palma de Mallorca, Spain
cllado@uib.es

Ramon Puigjaner
Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica
Ctra de Valldemossa, Km. 7.6, 07071
Palma de Mallorca, Spain
putxi@uib.es

## ABSTRACT

Performance model interchange formats are common representations for data that can be used to move models among modeling tools. In order to manage the research scope, the initial version of PMIF is limited to QNM that can be solved by efficient, exact solution algorithms. The overall model interoperability approach has now been demonstrated to be viable. This paper is a first step to broaden the scope of PMIF to represent models that can be solved with additional methods.

## General Terms

Performance, Simulation

## 1. INTRODUCTION

The Performance Model Interchange Format (PMIF) provides a mechanism for automatically moving queueing network performance models (QNM) among modeling tools. It was first introduced in 1995 [13] and later extended using XML as a viable mechanism for supporting model interchange [10]. Interchange formats have also been defined for layered queueing networks (LQN), UML, Petri Nets and other types of models. A framework has been developed to specify experiments to be solved, the output metrics to be gathered, and the transformation from output to useful results [11] [12].
The 2004 version of PMIF was limited to QNM that could be solved by efficient, exact solution algorithms to manage the scope of the research. The overall model interoperability approach has now been demonstrated to be viable. This paper is a first step to broaden the scope of PMIF to represent models that can be solved with additional methods.

## 2. QNM EXTENSIONS

The first step is to examine representative QNM tools, metamodels, and techniques to determine the features that should be supported. We examined features in:

Performance Engineering Book [9] - advanced model solution features that support Software Performance Engineering (SPE)

CSIM [2] - a powerful process-oriented simulation tool

Qnap [8] - a classic, full-featured QNM solver with both analytic and simulation solution capabilities

Java Modelling Tools (JMT) [4] - a recent QNM tool that incorporates features for modeling current systems

CSM/LQN [3] - a formal definition of the information requirements for Layered Queueing Networks

KLAPER [7] - a metamodel and language for evaluating system performance

These tools and techniques allow models to be solved with approximate analytical and/or simulation techniques. Table 1 shows a superset of features supported in these sources. The asterisks in the table indicate that it is possible to implement the feature using other features, but there is no primitive function provided.
This raises a key issue: ideally the PMIF extensions would include all of these features. However, the modern tools and techniques have higher level concepts such as messages and events while classic techniques and tools provide ways of implementing them indirectly. The PMIF extensions should support available features going forward, so we need a mechanism to address both the newer features and the classic ones.

## 3. PROPOSAL FOR PMIF EXTENSIONS

PMIF was based on concepts embodied in two earlier model interchange formats: the Electronic Data Interchange Format (EDIF) for VLSI designs [1] and the Case Data Interchange Format (CDIF) for software design interchange (also based on EDIF) [5]. Creators of EDIF envisioned the need to extend the model interchange formats (and thus the metamodels) and addressed it by providing for a concept of *levels* that add functionality at each successive level. Tools may

| Features | Book | CSIM | Qnap | JMT | CSM/LQN | KLAPER |
|---|---|---|---|---|---|---|
| Allocate | yes | RESERVE facility | yes | yes | Acquire | Acquire |
| Release | yes | yes | yes | no | Release | Release |
| Create passive server token | yes | use Event Set | yes | no | ? | no |
| Destroy (") | yes | use Event Clear | yes | no | ? | no |
| Create message token | yes | Mailbox Send | yes | no | Message | no |
| Destroy (") | yes | Mailbox Receive | yes | no | no | no |
| Create signal token | yes | Event Set | flag Set | no | no | no |
| Destroy (") | yes | Event Clear | flag Unset | no | no | no |
| Fork | yes | Create process(es) | yes | yes | yes | yes |
| Join | yes | WaitEvent | yes | yes | Merge | yes |
| Split | yes | Create process(es) | yes | fork | fork | fork |
| Phase change | yes | not needed | yes | no | not needed | not needed |
| Memory allocation | yes | STORE-ALLOCATE | * | no | Acquire + units | no |
| Memory release | yes | STORE-DEALLOCATE | * | no | Release + units | no |
| Memory add | yes | STORE- ADD | * | no | ? | no |
| External Resource | delay | use facility | yes | * | yes | * |
| Terminate | yes | yes | yes | no | no | no |
| Rerun-new simulation | no | yes | yes | no | no | no |
| Reset-counters in current run | no | yes | yes | no | no | no |
| Submodel | yes | * | yes | yes | no | no |
| Events | * | yes | yes | no | no | no |
| Mailbox or Message | * | yes | * | no | yes | yes |
| Compute | no | yes | yes | no | yes | yes |
| User-written subroutines | yes | yes | yes | no? | no | no |
| Interrupt | yes | no | yes | no | no | no |
| Get identity | no | yes | yes | no | no | no |
| Get-set priority | no | yes | yes | yes | ? | no |

**Table 1: Comparison of the features of QNM tools**

support different levels of the interchange format. The EDIF import philosophy is to import everything and for features that tools cannot handle to make appropriate substitutions. The extended version of PMIF can use levels to address the discrepancy in tools with higher level concepts and the classic features in other tools. So, the next level of PMIF will include those features common to most of the tools in Table 1. The next higher level will add the newer features in such a way that other tools will be able to import those models by mapping the features onto their own primitives. This step will be done in future work. Tools can continue to support a lower level of PMIF without change, or may opt to modify interfaces to support the additional functionality provided by extensions.

Other key differences in the tools and techniques are the supported arrival and service distributions and the queue scheduling disciplines. They vary so much that they are not included in the table.

Best practices in Service Oriented Architectures as defined by SOA Design Patterns [6] suggest generalizing the definition of context dependent settings such as these. In particular, the Validation Abstraction pattern suggests replacing constraints in metamodels and schemas with more general specifications. So, for example, rather than using an enumerated type with all of the queue scheduling disciplines explicitly defined, the pattern suggests defining it as a string. That allows tools to defer validation of the attribute when it is not necessary and it makes the evolution of the interchange formats easier because they do not have to be changed every time a new queue scheduling discipline is desired. The downside is that tools must be prepared to handle a situation when a feature is specified that the tool does not support. For example, if an unsupported queue scheduling discipline is specified, the tool could reject the model and return an error code, or just substitute another supported queue scheduling discipline and report the substitution.

The features in Table 1 above the double line are relatively easy to include in the first level of extensions. Figure 1 shows a revised PMIF meta-model. It adds a *SpecialServer* and *SpecialServiceRequest* specifications. The *Workloads* can be routed to the *SpecialServer* with normal *Transit* specifications. The *SpecialServiceRequest* provides specifications for Fork, Split, Join, Acquire, Release, Create, Destroy, Add, etc. behavior.

Figure 1 also adds *SolutionState* for simulation solutions. The *SolutionState* is based on the status of *Workloads* and *Nodes*. It may also include current confidence levels based on a variety of algorithms. *ControlSpecifications* such as Terminate, Rerun, and Reset cause changes in simulation behavior based on the solution state. *SpecialServiceRequests* and *Transits* may also depend on system state. For example, *Workloads* may be routed to the *Server* with the shortest queue. Similarly, phase changes depend on the current phase of the *Workload*.

The features in Table 1 below the double line are more difficult to represent. Events and Mailboxes require a mapping to classic tools. Compute statements, User-written subroutines, Get identity, etc. have no simple substitution for tools without these capabilities. These features will be addressed in future work.

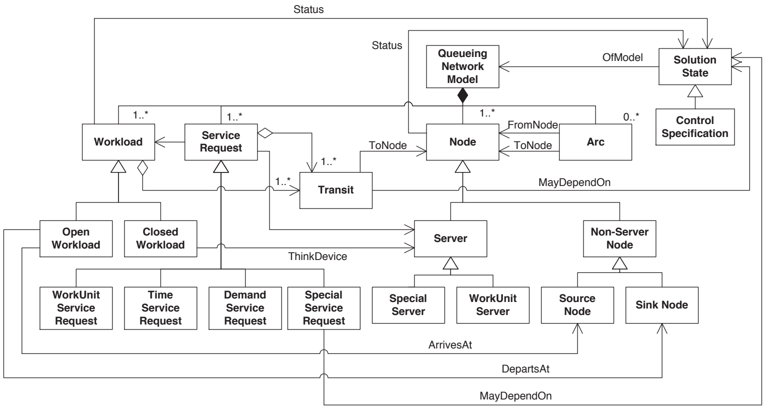

**Figure 1: PMIF Metamodel extension proposal**

## 4. CONCLUSIONS

This paper proposes extending the PMIF to relax the constraint that specified models must be solvable with efficient, exact solution algorithms. It presents a comparison of model features supported by a variety of representative tools and techniques. It adopts the concept of levels used in its predecessor EDIF and CDIF model interchange paradigms. A metamodel is proposed for the next level that represents many of the common model features in both modern and classic tools. The Validation Abstraction SOA Design Pattern is proposed so the future PMIF evolutionary changes will not require extensive changes to tool interfaces. Future work will refine this metamodel to include additional simulation control features, develop the schema and implement a proof of concept.

## 5. REFERENCES

[1] Edif users' group. design automation department.

[2] Mesquite software. `http://www.mesquite.com`.

[3] Puma project: Core scenario model. http://www.sce.carleton.ca/rads/puma/.

[4] M. Bertoli, G. Casale, and G. Serazzi. Jmt: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009.

[5] V. Electronics Industries Association, Arlington. Cdif - case data interchange format overview, eia/is-106, 1994.

[6] T. Erl. *SOA Design Patterns*. Prentice Hall, Upper Saddle River, NJ, 2009.

[7] V. Grassi, R. Mirandola, E. Randazzo, and A. Sabetta. Klaper: An intermediate language for model-driven predictive analysis of performance and reliability. pages 327–356, 2008.

[8] Simulog. Modline 2.0 qnap2 9.3: Reference manual, 1996.

[9] C. Smith. *Performance Engineering of Software Systems*. Addison-Wesley, 1990.

[10] C. Smith and C. Llado. 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.

[11] C. Smith, C. Llado, 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.

[12] C. Smith, LladóC.M., and R. Puigjaner. Automatic generation of performance results. 5652:73–78, July 2009.

[13] C. Smith and L. Williams. Panel presentation: A performance model interchange format. In *Proc. of the International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, 1995.