A Basic Performance Model Interchange Format

Connie U. Smith[†] and Lloyd G. Williams[§]

[†]Performance Engineering Services,
PO Box 2640, Santa Fe, New Mexico, 87504-2640 USA
Telephone (505) 988-3811, http://www.perfeng.com/~cusmith

[§]Software Engineering Research
264 Ridgeview Lane, Boulder, CO 80302
Telephone (303) 938-1147

May, 1997

This material is based upon work supported by the National Science Foundation under award number DMI-9361824. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

A Basic Performance Model Interchange Format

Abstract

This paper describes a mechanism to enable users of performance modeling tools to transfer models among a set of tools. This paper describes the evolution and definition of a basic PMIF. It describes how tool developers can implement the PMIF and how the export and import would work for users. A simple example illustrates the format. The paper describes how to extend the PMIF to add new features to the basic format. It ends with the current status of the PMIF, what we have learned, some suggestions for sets of extensions, and current work in progress.

1.0 Introduction

A performance model interchange format (PMIF) is a common representation for system performance model data that can be used to move models among modeling tools. A user of several tools that support the format can create a model in one tool, and later move the model to other tools for further work. For example, an analyst might create a model of a server platform conduct several studies, then move the model to a tool better suited to network analysis.

A PMIF would also permit modeling tool developers to solve test cases with a variety of tools to validate solution algorithms. It would let researchers of solution algorithms compare solutions from several sources. It would give tool vendors a relatively easy mechanism for exchanging models within their own product lines.

We are interested in tool inter-operability for Software Performance Engineering (SPE). SPE is a method for constructing software systems that meet performance goals [SMIT90a]. With SPE, developers build performance into systems rather than (try to) add it later. The SPE techniques use performance models to provide data for the quantitative assessment of the performance characteristics of software systems as they are developed. SPE has evolved over more than 15 years and has been demonstrated to be effective during the development of many large systems. Examples include [BELL88;BUT95;SMIT90a;SMI94d]. Despite SPE's documented successes it still faces barriers that hinder its widespread use. The principal problem is the gap between software developers who need the techniques and the performance specialists who have the skill to conduct performance engineering studies with most of today's modeling tools. Thus, extra time and effort is required to coordinate the design formulation and the design analysis. This limits the ability of developers to explore the design alternatives.

This work is part of a project to develop SPE tools to enable developers to conduct performance assessments of design alternatives. The ideal SPE tool will provide support for many SPE tasks in addition to the obvious requirement for performance modeling. The ideal SPE tool requires an adaptable approach that uses the performance modeling tool best suited to the software/hardware architecture issues and the life cycle stage of the assessment. It also requires a cost-effective solution that works with existing tools to the extent possible without requiring extensive changes to them. The performance model solution process should be transparent to the user of the SPE tool.

These requirements led to the initial version of the PMIF [SMI94b]. They also led to an SPE meta-model that defines the information requirements for SPE [WILL95]. The SPE meta-model can be used by developers of CASE tools who wish to collect data and provide for tool inter-operability with SPE tools to provide performance data for design analysis. It can also be used to exchange models among *software* performance modeling tools. Note that the PMIF is intended for *system* performance models that represent computer platforms and network interconnections with a network of queues and servers. The SPE meta-model, however, represents the software processing steps and other information for workloads that execute on the system performance model. This difference is explained in [SMIT90a;SMI96].

The remainder of this paper explains the PMIF and how to use it. The next section reviews its evolution: background and overview of the paradigm, and how we selected the elements of the PMIF. Next we present the current version of the basic PMIF. The basic PMIF has limited scope and is aimed at establishing the viability of the approach. Then we give an example and explain how developers can implement it and how the export and import would work for users. Then we describe how to extend the PMIF to represent additional model data, and review its current status, what we have learned since it was originally proposed, some suggestions for valuable extensions, and current work in progress.

2.0 PMIF Evolution

This section first reviews our choice for the PMIF paradigm, and briefly describes its components. Then it covers how we resolved inconsistencies in terminology and tool capabilities. It then explains the focus of the basic version of the PMIF and the approach for turning the basic version into a more general version after we establish its viability and future needs.

2.1 Paradigm Selected

We began this project by reviewing related research, the requirements for a PMIF and various notations for PMIF representation [SMI94b]. The VLSI community addressed similar requirements in the 1980s. They needed a mechanism for exchanging information about VSLI designs among chip foundries and research and production tools that conducted a variety of design analyses and error checking functions. They resolved the problems with a notation they called EDIF (Electronic Design Interchange Format). Later the computer-aided software engineering (CASE) community adapted the EDIF approach and developed CDIF (Case Data Interchange Format). They now have a set of proposed EIA/CDIF *standards* for exchanging various types of CASE information called CDIF *subject areas* [EIA94]. We determined that the benefits of adopting the proven EDIF/CDIF paradigm, which solves several of the difficult PMIF requirements, and the potential for eventually formulating both performance and SPE subject areas far outweighed the convenience of simpler notations such as a BNF grammar.

In the CDIF standard, the information to be transferred between two tools is known as a *model*. The contents of a model are defined using a *meta-model*. A meta-model defines the information structure of a small area of CASE (such as data modeling or data-flow diagrams) known as a “Subject Area.” Each meta-model is, in turn, defined using a *meta-meta-model*.[1] The meta-meta-model is based on the Entity-Relationship-Attribute (ERA) approach. The actual exchange of data uses a *transfer format* that is a straightforward derivation from the formal meta-model specifications. The CDIF transfer format uses a LISP format, so it is relatively easy to generate and interpret using LISP-based tools. LISP is the main drawback of the transfer notation; however the transfer format is meant to be machine processed and thus the human factors of the notation are less critical.

CDIF supports a wide range of designs and design notations through the subject areas. Extendibility is handled by defining new, named meta-models and transferring the meta-model definition along with the model transfer format. For example, if a tool wishes to export passive servers in the PMIF transfer format, it would include an extension section. The section would define the passive server entity, its attributes, and relationships to other entities, and the data for the passive servers in the performance model.

The CDIF meta-meta-model is used to define a PMIF meta-model. The PMIF meta-model defines a CDIF “Subject Area.” A CDIF Transfer Format then defines a standard format for transferring performance model information between tools.

We use a combination of the EIA/CDIF approach with object-oriented class diagrams. The graphical notation selected for this project is a subset of the OMT object-model notation developed by Rumbaugh, et.al. This notation includes graphical syntax for inheritance as well as for composite and associative entities and, thus, overcomes limitations we encountered with the EIA/CDIF graphical notation. The features of the OMT that we use maintain compatibility with the CDIF standard. The relevant elements of the OMT notation are defined in [RUMB91] and summarized in the appendix of [WILL95].

2.2 Terminology Issues

Performance model terminology varies considerably with modeling tools and textbooks on performance modeling. For example, the types of workloads are either described using the data processing terms of the workloads being modeled (e.g., timesharing, transaction, batch) or the queueing terms open and closed. Terminology choice often depends on the paradigm that the authors or developers select. Most terminology comes from either an operational analysis or a stochastic modeling paradigm. A taxonomy of current terminology is in [SMI94b]

There is not yet a consensus in the performance community on system performance model terminology. There is more agreement among the textbooks than the tools. This is probably because tool developers select terms that should be meaningful to

[1] **The terminology surrounding multiple layers of models, such as those used in the CDIF standard, can be confusing to the uninitiated. A model contains some information such as performance-model parameters. A meta-model is a model of the information contained in a model; i.e., it “models” or defines the model. A meta-meta-model is a model of the information contained in a meta-model; i.e., it is a general method for representing information requirements.**

the user who is focused on modeling a particular system rather than theoretical terms.

The PMIF must compromise among the options and must contain the information that can be translated into the other options. We have discovered that the specific terminology used in PMIF is less important than we originally thought. Tool developers are usually familiar with modeling theory and thus recognize that *class* and *workload* usually refer to the same thing; and *server* and *queue* are usually the same. Therefore, the meta-model described in the next section selects terms that should be meaningful to the developers of performance modeling tools. Tool users should not need to use the PMIF directly so the terminology does not need to be familiar to them.

2.3 Tool Compatibility Issues

The PMIF must be capable of expressing a wide range of models:

- those containing a small number of servers to very large numbers of servers
- from one to many workloads
- both open and closed models
- that may be solved using either analytic or simulation solution techniques

The PMIF must also be useful with existing tools. It must include modeling features that tools provide and support the modeling paradigms prevalent in today's tools. We examined the features supported by a representative set of performance modeling tools. This section summarizes the results of a taxonomy of these tools (see [SMI94b] for the taxonomy).

The tools selected for the taxonomy are representative of the types of products currently available. The products support system and software models as well as custom models that may model more general systems in addition to computer systems. Thus, they are the types of products that we wish to interchange model information among. The following characteristics of the tools are pertinent to PMIF choices:

- The tools use a range of solution types: analytic, simulation, and hybrid. Different tools require different additional information. The analytic tools support a subset of all models; the simulation tools can use additional model information.
- Modeling tools originally had language-oriented user interfaces. Now, many support graphical user interfaces (GUI), spreadsheet-type interfaces, and forms-based interfaces.
- There are three basic approaches to specifying service requests for servers in the model:
  - Time: specifies the service time per visit to a server
  - Demand: specifies the total service time for all visits to a server
  - Computed: calculates either time or demand from other data specified in the model.

- Most tools model time units implicitly: all specifications must be in the same relative time units (e.g. sec. or hours) the modeler can choose the time unit most convenient for the problem. Some tools let the user choose a time unit for each specification; others use a time unit appropriate for the device or specification, such as ms. for disk devices and transactions per hour for an arrival rate.
- Routing specifications vary considerably. Some tools specify routing among servers by specifying the probability that a job goes from device A to device B. Analytic tools that specify demand service requests do not need routing because they specify the total demand for all visits. One tool specifies routing through logic in C code, and others compute routing from other specifications.
- Queue-scheduling disciplines (QD) also vary among tools. Simulation based tools generally support many more QDs than analytic tools. “Standard” QD for analytic tools are those that have efficient, exact analytic solutions (FCFS, PS, and IS). Many support priority scheduling with analytic approximations.
- “Domain” is IBM / MVS terminology for a feature that limits the number of jobs in the system by workload. Some analytic tools support this with an approximate analytic solution. Others have no support for “domains”. Simulation tools can support the concept through primitives that let the modeler simulate the holding of jobs when the number in the system exceeds the limit.

2.4 Basic PMIF

The full PMIF is beyond the scope of this initial investigation. This paper presents a basic PMIF. It addresses a specific type of performance model: Queueing Network Models (QNM) that may be solved using exact, analytic solution algorithms [JAIN90;MENA94;MOLL89]. This basic version of the PMIF demonstrates the feasibility of the approach by addressing a subset of the full PMIF. It seems best to have the kernel of the PMIF support a minimum content set and use CDIF meta-model extensions to add feature sets supported by a subset of the tools. This basic PMIF will represent the subset of data needed by all the analytic tools.

CDIF provides capabilities for transferring graphical information along with the model. An importing tool can import the graphics information but use its own icons for the various model components. The specific mechanism for specifying the graphical view of the model eventually must be handled by the PMIF; however the visual representation of models is not a central issue in establishing PMIF feasibility. Therefore, this feasibility study omits graphics. The additional extensions for visual representation are a topic for future research. Features supported by some tools, such as priority scheduling and domains, that can be solved with *approximate* solution algorithms will also need to be added.

We envision the use of the EDIF concept of *levels* in the PMIF. Level 0 contains elements needed by all tools. Level 1 adds features needed by most but not all tools. Subsequent levels add tool-specific features, research and development features, etc. So level 0 PMIF would be the basic PMIF, level 1 adds features in common QNM approximations, and level 2 adds simulation features. Other features, such as layered queueing networks [Woodside ref still needed], would be special extensions.

There is currently no consensus on the service time versus demand and transition probability versus server visits. PMIF will support either demand or time service requests. It specifies routing through visits (or demand) because most analytic tools use this approach. The other tools can convert visits to probabilities when they import the PMIF model. It does not include results; further research is needed to integrate them into the meta-model.

3.0 QNM Meta-model

This model is known as the QNM meta-model because it is a model of the information that goes into constructing a QNM. This meta-model serves two purposes. The first is to provide a rigorous definition for the information required for a QNM that may be solved using exact analytical techniques. This is valuable to performance tool vendors because it defines the information that may be exported and imported between tools that support this initial version of a PMIF.

The second purpose of the meta-model is to generate a prototype version of a formal PMIF using the CDIF transfer format derived from the meta-model. With the PMIF, performance modeling tools can exchange information, and SPE tools can use a performance modeling tool best suited to the software/hardware architecture issues and the life cycle stage of the assessment.

This section begins with a textual description of the QNM meta-model. A discussion of the issues discovered in the creation of the meta-model and how they were resolved in this initial version follow it.

3.1 QNM Meta-Model Description

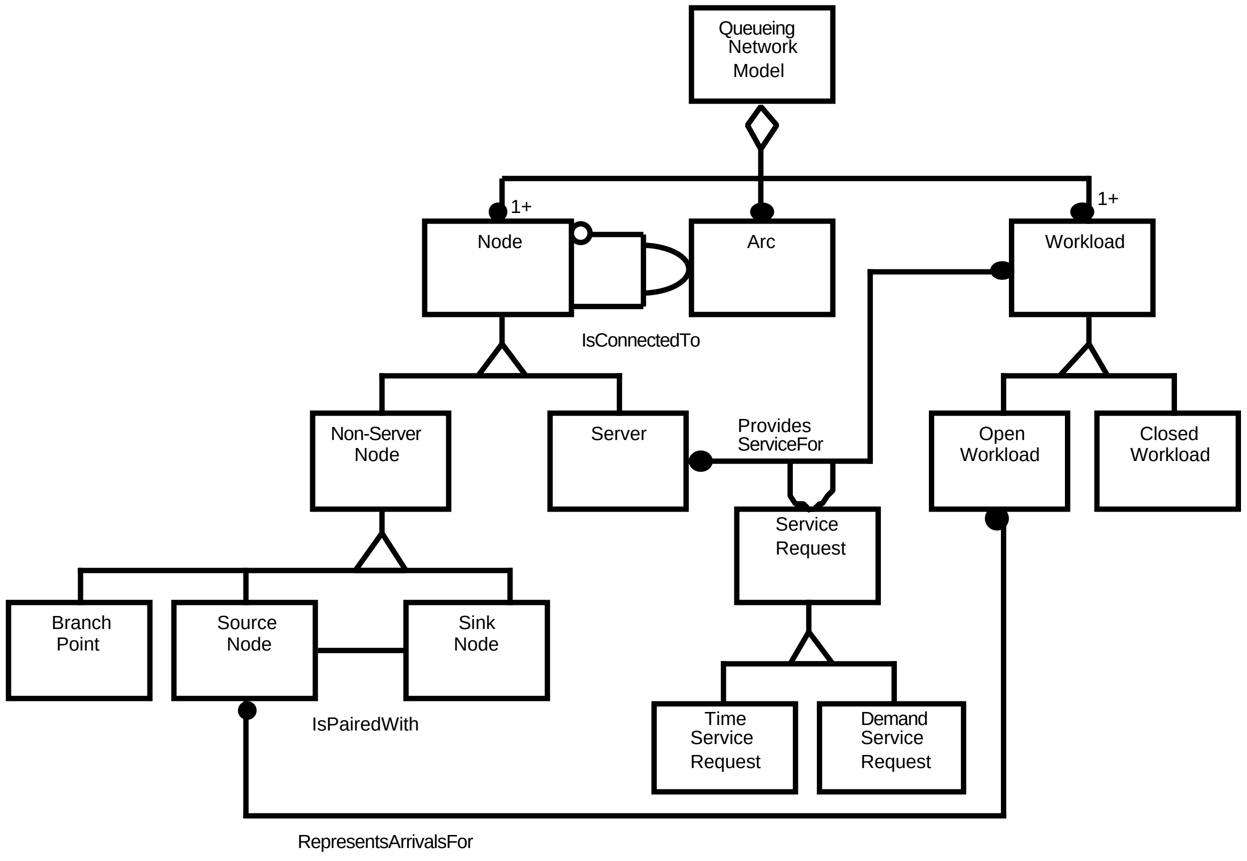
The meta-model Entity-Relationship-Attribute (ERA) diagram is shown in Figure 1. The ERA notation with the OMT extensions is described in [WILL95] The following paragraphs describe the entities and their relationships.

A *QueueingNetworkModel* is composed of one or more *Nodes*, zero or more *Arcs*, and one or more *Workloads*. An *Arc* connects one *Node* to another *Node*. Several types of *Nodes may* be used in constructing a *QueueingNetworkModel*:

- *Server*: represents a component of the execution environment that provides some processing service.
- *Non-ServerNode* represents connectivity in the model, but the nodes do not provide service. There are three types of *Non-ServerNodes*
  - *SourceNode:* represents the origin of an *OpenWorkload*.
  - *SinkNode* represents the exit point of an *OpenWorkload*.
  - *BranchPoint* is a convenient way to represent the origin or destination of multiple arcs.

A *Server* provides service for one or more *Workloads*. A *Workload represents* a collection of transactions or jobs that make similar *ServiceRequests* from *Servers* in the *QueueingNetworkModel*. There are two types of *Workloads:*

Figure 1. Queueing Network Meta-Model



The following shows the entities in the above diagram and their attributes. Note that EIA/CDIF supports inheritance. For example, DemandServiceRequest inherits the attributes from ServiceRequest, so in addition to ServiceDemand it also has the inherited attributes WorkloadName, ServerID, and TimeUnits.

- Arc
  - Description
  - FromNode
  - ToNode
- BranchPoint
- ClosedWorkload
  - NumberOfJobs
  - ThinkTime
  - TimeUnits
- DemandServiceRequest
  - ServiceDemand
- Node
  - Name
  - ID
- Non-ServerNod e
  - NodeType
- OpenWorkload
  - ArrivalRate
  - TimeUnits
- QueueingNetworkModel
  - Name
  - Description
- Server
  - Quantity
  - SchedulingPolicy
- ServiceRequest
  - WorkloadName
  - ServerID
  - TimeUnits
- SinkNode
- SourceNode
- TimeServiceRequest
  - ServiceTime
  - NumberOfVisits
- Workload
  - WorkloadName

- *OpenWorkload:* represents a workload with a potentially infinite population where transactions or jobs arrive from the outside world, receive service, and exit. The population of the *OpenWorkload* at any one time is variable.
- *ClosedWorkload:* represents a workload with a fixed population that circulates among the *Servers*.

A service request associates the *Workloads* with *Servers*. A *ServiceRequest* specifies either the average *TimeService* or *DemandService* provided for each *Workload* that visits the *Server*. A *TimeServiceRequest* specifies the average service time and number of visits provided for each *Workload* that visits the *Server*. A *DemandServiceRequest* specifies the average service demand (service time multiplied by number of visits) provided for each *Workload* that visits the *Server*.

The complete EIA/CDIF specification for the PMIF is in [SMI94b]. It formally defines the entities, each of the attributes, and the relationships.

The EIA/CDIF transfer format is derived from the meta-model. It is a Lisp-style notation. Each of the entities and its attributes is represented as follows:

```
(EntityName EntityID
  (Attribute1 Attribute1Value)
  …
  (AttributeN AttributeNValue)
)
```

For example, an arc entity in a queueing model may be defined as:

```
(Arc QNM001.1
  (FromNode #d4)
  (ToNode #d3)
)
```

Arc is the EntityName and the EntityID is the Meta-entityID followed by an InstanceNumber, i.e., QNM001.1. The arc has no (optional) description, and specifies the FromNode is nodeID 4 and ToNode is nodeID 3. The notation #d indicates a decimal value; the value is 4 from the FromNode id. Section 4 demonstrates the transfer format for an example.

3.2 Meta-Model Issues

This meta-model proposes the following compromises on terminology and features:

- Visits vs. probabilities: We propose using the operational analysis term *visits* rather than the stochastic modeling *probability*. Fewer tools use probabilities and they can calculate probabilities from visits.
- Demand vs. service time: We propose both – a ServiceDemand may be either a TimeServiceRequest or a DemandServiceRequest.
- Queue scheduling disciplines: QD is an attribute of Server. We propose an enumerated type that initially supports the “standard” set (Processor Sharing, First-Come-First-Served, and Infinite Server). Extensions will be straightforward.
- Workload types: We propose OpenWorkloads and ClosedWorkloads. These terms are familiar to tool developers.

Several entities and relationships are not required for this basic PMIF, but are included for future expansion:

- Arcs are not needed if visits or demand is specified. They will be needed for the graphical representation of the model but, semantically, they represent more than graphical elements for simulation solvers. We explicitly include them in the basic PMIF to have the proper foundation for anticipated extensions.
- Non-server nodes are not needed for the efficient, exact solution subset. They will be needed for the graphical representation and for simulation models.
- The relationship between the open workloads and the source node is not needed for this model subset. It will be needed for simulation models.

4.0 How to use PMIF

An example based on a simple automated teller machine (ATM) illustrates the definition of a QNM with the proposed PMIF. This example illustrates using the PMIF for SPE. We mentioned earlier that PMIF has more general applicability, but we are particularly interested in using a variety of system modeling tools in conjunction with an SPE tool. First we illustrate the PMIF transfer format for the example. Then we describe the PMIF import and export philosophy inherited from EIA/CDIF.

4.1 ATM Example

The ATM accepts a cash card and requests a personal identification number (PIN) for verification. Customers can perform any of three transactions at the ATM: deposit cash to an account, withdraw cash from an account, or request the available balance in an account. A customer may perform several transactions during a single ATM session. The ATM communicates with a computer at the host bank, which verifies the account and processes the transaction.

The example models two workloads as illustrated in Figure 2. One is an ATM session in which a customer requests a Withdrawal transaction (on the left). The other is an ATM session in which a customer requests a Get balance transaction (on the right). Each of the workloads is depicted with an *execution graph* software model. The model creation and solution is described in detail in [SMIT90a]; this section summarizes the modeling results. The software model solution yields the following resource requirements for the servers in the model:

- Withdrawal: (630, 11, 8)
- Get Balance: (250, 6, 3)

for the CPU, ATM, and DEV1 respectively.

Note that the case study picture shows three disk devices (DEV1 - DEV3). The software models only use DEV1, so it is the only one illustrated with the PMIF.

Figure 2. Case Study Model

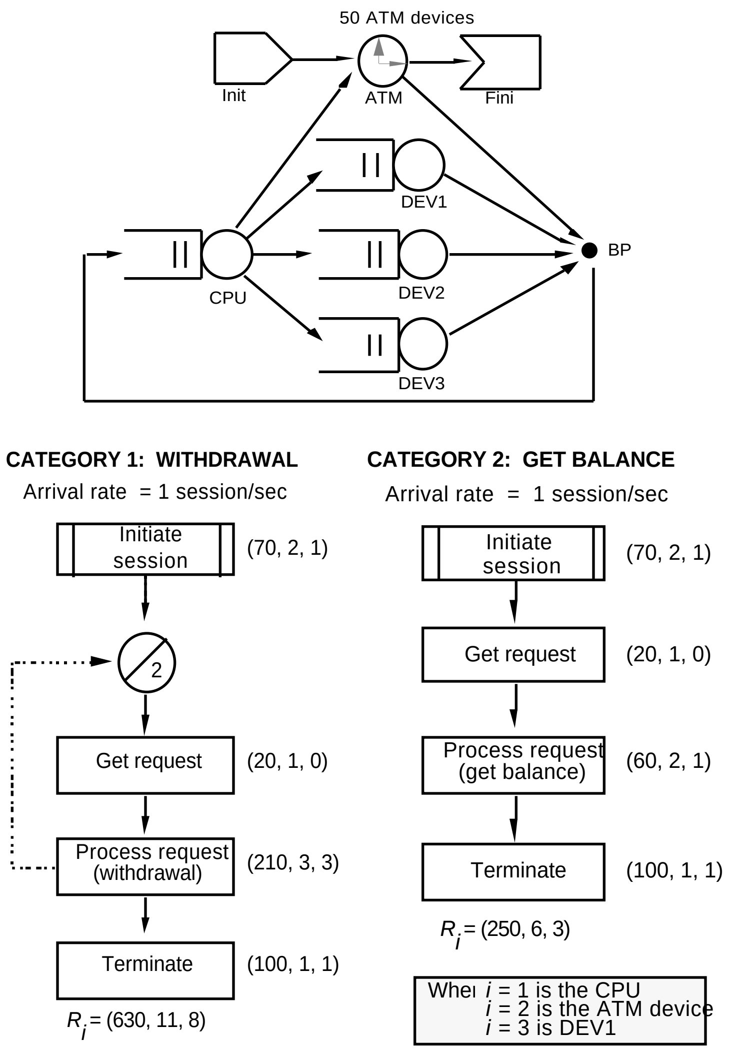


Table 3. QNM Parameters

Nodes	Visits (Withdraw)	Visits (Get Bal)	Service Time (Withdraw)	Service Time (Get Bal)
Init	1	1		
ATM	11	6	1	1
Fini	1	1		
DEV1	8	3	0.05	0.05
BP	18	8		
CPU	all	all	0.00630	0.00250

These resource requirements are translated into the parameters for the QNM that are shown in Table 3. Note also that the values for the CPU differ from the model parameters in Table 5.6 in [SMIT90a]. However this case study uses the total demand for the CPU instead to demonstrate the DemandServiceRequest.

So, for this SPE example, an analyst first collected the SPE information and created a *software model* for each of the two performance scenarios (workloads). The software models were solved using graph analysis algorithms. The model solution yields the resource requirements for the computer devices for each scenario. SPE information about the envisioned computer environment produced the *system model* topology and the service rates for the devices in the model shown at the bottom of Figure 2. This is combined with the results of the software model solution to produce the parameters for the system model. Next, we translate this system model (QNM) into the PMIF format.

4.2 ATM Example PMIF

The PMIF format is the transfer format derived from the QNM meta-model using the EIA/CDIF standards. The complete model is in the Appendix. It shows how to represent the data in Table 3 in the PMIF format. This section describes some key portions of the PMIF.

A Workload in the model is defined as follows:

```
  (OpenWorkload QNM007.1
    (WorkloadName “Withdrawal”)
    (ArrivalRate #d1)
    (TimeUnits <Sec>)
  )
```

The case study has two open workloads; each has an arrival rate of 1 session per second.

A Server in the model is defined as follows:

```
  (Server QNM009.1
    (Name “CPU”)
    (ID #d1)
    (Quantity #d1)
    (SchedulingPolicy <PS>)
  )
```

This specification defines one CPU server (with id number 1) with a Processor Sharing queue-scheduling discipline. Three servers are defined in the case study: CPU, ATM and DEV1. The case study also has three Non-serverNodes: the Source node “Init”, the Sink node “Fini”, and a BranchPoint “BP”. Their definition is in [SMI94b]; the data is in the appendix.

A ServiceRequest for the CPU is specified as follows:

```
  (DemandServiceRequest QNM004.1
    (WorkloadName “Withdrawal”)
    (ServerID #d1)
    (TimeUnits <Sec>)
    (ServiceDemand #d0.00630)
  )
```

This specifies that the Withdrawal workload makes a total demand of 0.00630 seconds at the CPU server.

A ServiceRequest for the disk device is specified as follows:

```
  (TimeServiceRequest QNM013.1
    (WorkloadName “Withdrawal”)
    (ServerID #d2)
    (TimeUnits <Sec>)
    (ServiceTime #d0.05)
    (NumberOfVisits #d8)
  )
```

This specifies that the Withdrawal workload makes 8 visits to DEV1, each visit requires an average of 0.05 sec.

The PMIF in the appendix also shows the arc specifications and the IsPairedWith and RepresentsArrivalsFor relationships. The other relationships depicted in the QNM meta-model (eg. IsConnectedTo, ProvidesServiceFor, etc.) are explicitly described with associative entities (e.g. Arcs and ServiceRequests, etc.) therefore their relationship declaration is unnecessary.

4.3 Import and Export Philosophy

To *export* models with PMIF, tools provide all data they have that is specified in the meta-model. Tools must provide default values for the essential data in the PMIF meta-model if other values are not available. For example, analytic modeling tools may implicitly assume a queue scheduling discipline. They should provide a default specification, such as processor sharing, in the PMIF. If they wish to provide additional data that is not specified in the basic model, such as priorities of workloads, or service time distributions, they provide the meta-model extensions defining the additional model data and the data itself. Extensions are covered in the next section.

To *import* models in the PMIF format, tools use the data provided, discard data items they do not need, and make assumptions about data items not in the basic meta-model that they require. For example, tools might ignore meta-model extensions for which they have other defaults, such as service time distributions. They may assume equal priorities for workloads because priorities are not part of the basic meta-model.

The users of the tools will likely need to provide additional information to the receiving. tool. If they are using the destination tool to study additional facets of system performance, the corresponding data was probably not in the originating tool.

5.0 PMIF Extensions

As discussed earlier, the PMIF meta-model presented here is a preliminary version. It is likely that, as the PMIF is used, the current version will need extensions to provide comprehensive support. There are two possible mechanisms for extending the PMIF.

The first mechanism is to extend the meta-model itself. This approach is appropriate for QNM features that would be applicable to most if not all tools that would use the interchange format. The meta-model can be extended in several ways, depending on the nature of the change. The simplest extension is the addition of an attribute to a meta-entity. For example, this approach might be used to add priorities to workloads by adding a Priority attribute to the Workload entity in the Meta-Model ERA diagram. More complex extensions might involve adding new meta-entities, either as stand-alone entities or as subclasses of existing meta-model entities.

The second mechanism makes use of the CDIF extensibility facility. This mechanism allows two or more tools to share information that is not part of a CDIF meta-model. A tool that exports the extra information simply provides definitions for the new information before transferring the information itself. An importer must be able to parse the information but does not need to do anything with it. Thus, if an exporter provides non-standard information as part of a CDIF transfer file, the importer will only use that information if it can. This approach is most useful for features that are unique to one or, at most, a few tools, such as Layered QNMs [Same Woodside ref needed].

Note that the ATM example showed that a software model was first solved to produce a QNM to be exported. The resulting PMIF did not contain details of the software performance model. Some tools wish to exchange information about the software model in addition to the system model. The best way to handle software model data interchange is to use the SPE meta-model [WILL95] with a transfer format derived from it, rather than adding software model extensions to the PMIF.

6.0 Status

The original proposal was the result of a research project. It was distributed to tool developers who volunteered to review it. Several good suggestions were made and incorporated into the meta-model. The resulting proposal was presented to the performance community [SMI95b;SMI95c]. So far, tool developers and performance analysts generally agree with adopting the CDIF paradigm for expressing the meta-model. They also agree with the terminology defined in the model. Some disagree with the simplicity of the model and would like extensions included in the baseline model.

In the next version of the PMIF, we probably want a third type of ServiceRequest that specifies a ServiceUnit (such as 50 ms. per I/O) for a Server then has a UnitServiceRequest that specifies the number of units of service requested (such as 11 I/Os). A few tools currently use this feature, it is desirable for SPE models, and participants at the “Heidelberg Tools Conference” want its conceptual representation benefits. In particular, they want to separate the *work* (e.g. the number of I/Os) from the *machine* that provides the service (the time per I/O).

A future version will also include a visual representation of the model. The EIA/CDIF approach has a separate subject area to represent the picture of the model because there are so many graphical notations for software designs. With the CDIF approach, the interchange of the graphical information for a performance model would use two different meta-models: one for the model parameters, and another for the model picture. Queueing network model pictures have a general form even though the exact symbols for nodes differ slightly. We could adopt the CDIF paradigm, however, it appears that the QNM picture could be handled with additional attributes attached to nodes to define “a virtual grid” location, and a picture-element type. Then tools could use their own node symbols and transform the virtual grid location into device coordinates appropriate for their system.

The best and most extensive suggestion was to incorporate specifications for an experimenter component that specifies model studies to be run, variations in parameter settings, etc. Some researchers believed that adding this expressive power would require modifications to the basic meta-model. After reviewing the requirements, it appears that it is possible to handle the experimenter specifications as a meta-model extension, and to tie the model results to the experimenter rather than the QNM meta-model. Further research is required to determine the best way to handle this facet.

There is interest in refining the PMIF contents, and supporting it in many performance modeling tools. Many have volunteered to participate in developing a standard for performance model interchange that will be supported by a variety of performance modeling tools.

The next step is to implement a prototype and learn whether the proposed version is viable for tools with very different features. We are actively working with several tool developers and researchers to do this. (Note to reviewers: we hope to have results to discuss at the conference).

7.0 Summary

This paper describes a basic Performance Model Interchange Format (PMIF) for exchanging Queueing Network Models (QNMs) among performance modeling tools. It would permit tool users to create a model with one tool, conduct studies, and export the model to another tool better suited to different types of performance studies. The basic PMIF is based on the representation technique of the EIA/CDIF standard. It consists of a QNM meta-model depicted with an extended ERA diagram and the formal definition of the entities, relationships and attributes in the model. The transfer format derived from the QNM meta-model serves as the PMIF representation.

The PMIF content represents features from a representative set of today’s modeling tools. The basic PMIF defined here represents the subset of data needed by all the analytic modeling tools. Features supported by some tools, such as priority scheduling and domains can be added using meta-model extensions. Terminology used in the PMIF is based on terms that are meaningful to developers of performance modeling tools rather than terms familiar to tool users.

The PMIF is defined and used in an example. We also explained techniques for using and extending the PMIF. The paper demonstrates the feasibility of defining a QNM in a standard format that will permit models defined in the format to be solved by all tools that support the format.

This paper describes the latest version of the PMIF. It incorporates modifications and extensions based on feedback from researchers and developers of performance modeling tools. We hope that it will be implemented and used by a variety of performance modeling tools.

Several extensions are needed: the graphical representation of the model, the experimenter, the studies, the results, and additional model features such as priorities, domains, passive resources, scheduling disciplines, and so on. Our overall goal is to support software design decision analysis through the smooth (transparent) transfer of information from software designs to performance model solutions and back to the designer. The PMIF facilitates additional performance modeling studies.

References

[BELL88] Thomas E. Bell, ed., *Special Issue on Software Performance Engineering*, , 1988.

[BUT95] Janet Butler, ed., *Software Performance Engineering for Client/Server*, 15, 10, Applied Computer Research, Phoenix, 1995.

[EIA94] EIA, CDIF - CASE Data Interchange Format Overview, 1994.

[JAIN90] R. Jain, *Art of Computer Systems Performance Analysis*, John Wiley, New York, NY, 1990.

[MENA94] Daniel A. Menascé, Virgílio A.F. Almeida, and Larry W. Dowdy, *Capacity Planning and Performance Modeling*, PTR Prentice Hall, Englewood Cliffs, NJ, 1994.

[MOLL89] Michael K. Molloy, *Fundamentals of Performance Modeling*, MacMillan, 1989.

[RUMB91] J. Rumbaugh, et al., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[SMIT90a] Connie U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, Reading, MA, 1990.

[SMI94b] Connie U. Smith, Definition of A Performance Model Interchange Format, 1994.

[SMI94d] Connie U. Smith and Bernie Wong, "SPE Evaluation of a Client/Server Application," *Proc. Computer Measurement Group*, Orlando, FL, Dec., 1994, .

[SMI95b] Connie U. Smith, "SPE Experience and Recent Research," *Featured talk at the SEI Workshop on Effective Practice in Performance Engineering*, Pittsburg, PA, June, 1995, 1995, .

[SMI95c] Connie U. Smith and Lloyd G. Williams, "Panel Presentation: A Performance Model Interchange Format," *International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, Heidelberg Germany, September, 1995.

[SMI96] Connie U. Smith, "Designing High-Performance Distributed Applications Using SPE: A Tutorial," *Proc. Computer Measurement Group*, San Diego, CA, Dec., 1996, 498-507.

[WILL95] Lloyd G. Williams and Connie U. Smith, "Information Requirements for Software Performance Engineering," *Proceedings 1995 International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, Heidelberg, Germany, September, 1995.

Appendix: PMIF Transfer Format

```
#| Model Section |#
(:MODEL
  (QNM Model QNM008.1
    (Name "ATM Sample")
  )
  (OpenWorkload QNM007.1
    (WorkloadName "Withdrawal")
    (ArrivalRate #d1)
    (TimeUnits <Sec>)
  )
```

```
  (OpenWorkload QNM007.2
    (WorkloadName "Get Balance")
    (ArrivalRate #d1)
    (TimeUnits <Sec>)
  )
  (Server QNM009.1
    (Name "CPU")
    (ID #d1)
    (Quantity #d1)
    (SchedulingPolicy <PS>)
  )
  (Server QNM009.2
    (Name “DEV1”)
    (ID #d2)
    (Quantity #d1)
    (SchedulingPolicy <FCFS>)
  )
  (Server QNM009.3
    (Name “ATM”)
    (ID #d3)
    (Quantity #d50)
    (SchedulingPolicy <IS>)
  )
  (SourceNode QNM012.1
    (Name “Init”)
    (ID #d4)
    (NodeType <Source>)
  )
  (SinkNode QNM011.1
    (Name “Fini”)
    (ID #d5)
    (NodeType <Sink>)
  )
  (BranchNode QNM011.1
    (Name “BP”)
    (ID #d6)
    (NodeType <Branch>)
  )
  (DemandServiceRequest QNM004.1
    (WorkloadName “Withdrawal”)
    (ServerID #d1)
    (TimeUnits <Sec>)
    (ServiceDemand #d0.00630)
  )
  (TimeServiceRequest QNM013.1
    (WorkloadName “Withdrawal”)
    (ServerID #d2)
    (TimeUnits <Sec>)
    (ServiceTime #d0.05)
    (NumberOfVisits #d8)
  )
  (TimeServiceRequest QNM013.2
    (WorkloadName “Withdrawal”)
    (ServerID #d3)
    (TimeUnits <Sec>)
    (ServiceTime #d1)
    (NumberOfVisits #d11)
  )
  (DemandServiceRequest QNM004.2
    (WorkloadName “Get Balance”)
    (ServerID #d1)
    (TimeUnits <Sec>)
    (ServiceDemand #d0.00250)
  )
  (TimeServiceRequest QNM013.1
    (WorkloadName “Get Balance”)
    (ServerID #d2)
    (TimeUnits <Sec>)
    (ServiceTime #d0.05)
    (NumberOfVisits #d3)
  )
  (TimeServiceRequest QNM013.2
    (WorkloadName “Get Balance”)
    (ServerID #d3)
    (TimeUnits <Sec>)
    (ServiceTime #d1)
    (NumberOfVisits #d6)
  )
  (Arc QNM001.1
    (FromNode #d4)
    (ToNode #d3)
  )
  (Arc QNM001.2
    (FromNode #d3)
    (ToNode #d5)
  )
  (Arc QNM001.3
    (FromNode #d3)
    (ToNode #d6)
  )
  (Arc QNM001.4
    (FromNode #d6)
    (ToNode #d1)
  )
  (Arc QNM001.5
    (FromNode #d1)
    (ToNode #d2)
  )
  (Arc QNM001.6
    (FromNode #d1)
    (ToNode #d3)
  )
  (Arc QNM001.7
    (FromNode #d2)
    (ToNode #d6)
  )
  (IsPairedWith QNM036.1 QNM006.1 QNM006.2)
  (RepresentsArrivalsFor QNM037.1 QNM006.1  QNM007.1)
  (RepresentsArrivalsFor QNM037.2 QNM006.1  QNM007.2)
)
```