Appears in Proc. QEST 2004.

# Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation

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

Catalina M. Lladó
*Universitat Illes Balears*
*Departament de Matemàtiques i Informática*
*Carretera de Valldemossa, Km. 7.6,*
*07071 Palma de Mallorca, Spain*
*cllado@uib.es*

## Abstract

*A Performance Model Interchange Format (PMIF) provides a mechanism whereby system model information may be transferred among performance modeling tools. The PMIF allows diverse tools to exchange information and requires only that the importing and exporting tools either support the PMIF or provide an interface that reads/writes model specifications from/to a file. This paper presents a new version of the PMIF specification (PMIF 2.0) and its XML implementation. The paper also describes a prototype that was implemented to prove the concept, in which the exporting tool is SPE·ED and the importing tool is Qnap, and it discusses the issues in this and the reverse exchange. It shows the validation of the prototype based on the solution of examples that were exported from SPE·ED and imported by Qnap. In addition, it proposes some extensions to PMIF 2.0.*

## 1. 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 without the need to laboriously translate from one tool's model representation to the other. For example, an analyst might create a model of a server platform to conduct several studies, then move the model to a tool better suited to network analysis. Other uses for the PMIF include enabling users to:

1. compare solutions from multiple tools.
2. create input specifications in PMIF or in a familiar tool rather than learn the interface to multiple tools.
3. migrate a model to temporarily use another tool to study more detailed models.
4. migrate a model to permanently use a different tool.
5. create software performance models to study architecture and design trade-offs, then use another tool to study details of the computer system.
6. compare different tools before buying one.

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

PMIF provides a common interface to tools. Without it two tools would need to develop a custom import and export mechanism. A third tool would require a custom interface between each of those tools resulting in a $2^N$ requirement for customized interfaces. With PMIF, tools export and import with the same format so the requirement for customized interfaces is reduced to $2 \cdot N$.

With XML (Extensible Markup Language) tools the complexity and amount of effort to create the PMIF interface is quite small [13]. While XML is verbose, PMIF is a course-grained interface. A file is exported, sent to another tool, it is imported and the model solved. So the performance impact of XML as the interface is insignificant compared to a fine grained interface that exchanges each XML element as it is generated.

Earlier work defined a PMIF using an EIA/CDIF (Electronic Industries Association/CASE Data Interchange Format) paradigm that calls for defining the information requirements for a Queueing Network Model (QNM) with a meta-model [8-10]. A transfer format was then created from the meta-model and used to exchange information.

This project uses that work (PMIF 1.0) as a starting point, updates the meta-model with information deemed to be necessary during this implementation, then specifies an XML schema for the resulting PMIF 2.0 meta-model. We implemented a prototype export mechanism from the SPE·ED software performance modeling tool [6] into pmif.xml, and a prototype import mechanism from

pmif.xml into the Qnap system performance modeling tool [7]. Qnap is a modeling tool that can be used on its own or through Modline, which provides a graphical user-friendly interface for the model definition and interactive visualization of results, among others, using Qnap to solve those models. We used the prototypes to study several examples. Our use of unlike tools helped us find limitations in the meta-model and find a general way to resolve them. The example solutions confirm that the pmif.xml transfer was successful. The examples provide a set of models that are well documented, with reproducible results, that may be used by others who wish to explore the pmif.xml approach to interchanging models.

There has been other related work in this general area. For example, Coretellessa and Mirandola annotate UML diagrams and transform them into Execution Graphs and Queueing Network Models [3]. More recently, 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 [2]. Gu and Petriu use XSLT (eXtensible Stylesheet Language for Transformations) [13] to transform UML models in XML format to the corresponding Layered Queueing Network (LQN) description which can be read directly by existing LQN solvers [5]. Wu and Woodside use an XML Schema to describe the contents and datatypes that a Component-Based Modeling language (CBML) document may have [15]. CBML is an extended version of the Layered Queuing Network (LQN) language that adds the capability to model software components and component based systems. These works 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.

This paper first summarizes the PMIF paradigm, its development and the current version of the PMIF meta-model. Section 3 covers the XML implementation and the modifications that were required for PMIF 2.0. Section 4 describes the resulting XML schema. Section 5 describes the prototypes and issues in exporting and importing pmif.xml between the two tools. Section 6 covers the examples. The paper also reports on the status, some extensions and future work, and conclusions. Many details that would not fit in this limited space are in [12]. (Reader, we are sorry for the inconvenience.)

## 2. PMIF summary

PMIF 1.0 was based on EIA/CDIF [4], a family of standards for transferring information between CASE tools. An exchange takes place via a file and internal tool information is translated to and from the file's transfer format. The CDIF paradigm was used to define a QNM meta-model [10]. The transfer format in the original CDIF standard used LISP as the implementation language. Today, XML is a more logical choice for a transfer format because it was designed for this purpose and there are many tools available to support the exchange of information in XML.

This work uses the PMIF 1.0 meta-model as a starting point because it is a good description of the information requirements for performance model interchange, but uses XML to implement the transfer format.

The contents of the PMIF 1.0 Meta-Model resulted from a taxonomy of the terminology used for QNM in performance tools and performance textbooks, and of the features provided by available tools for solving performance models [10]. A wide variety of features and terms were considered, as well as feedback from researchers in the performance field.

PMIF 1.0 established the viability of the approach. It started with a meta-model of the information required for a manageable QNM subset: the data needed for QNM that may be solved using exact analytic solution algorithms. We discuss mechanisms for extensions later.

This work also begins with PMIF 1.0 as a starting point to establish the viability of using XML as the transfer format. During the implementation of the XML transfer, we discovered some modifications that were required to exchange the general QNM among unlike tools. The next section describes the modified meta-model.

## 3. QNM Meta-model 2.0

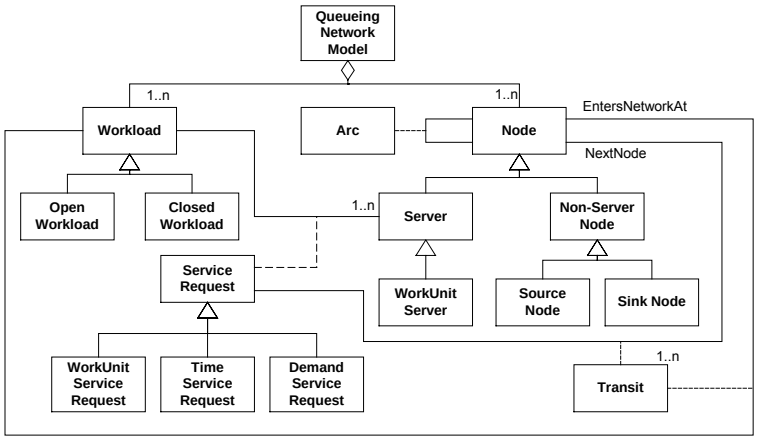
### 3.1. QNM Meta-model 2.0 description

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: To provide a rigorous definition for the information required for a QNM that may be solved using exact analytical techniques; and to generate the formal PMIF using the XML transfer format derived from the meta-model.

The meta-model diagram in Figure 1 is based on the PMIF meta-model defined in [10]. This version uses UML, adds information, and eliminates other information. Subsequent sections provide more details on the content changes in version 2.0. The following is an abbreviated description of the diagram. Complete details are in [12].

A *QueueingNetworkModel* is composed of one or more *Nodes*, and one or more *Workloads*. A *Server* provides service for one or more *Workloads.* A *Workload* represents a collection of transactions or jobs that make similar *ServiceRequests* from *Servers*. There are two types of *Workloads: OpenWorkload* and *ClosedWorkload.*

**Figure 1. Queueing Network Meta-Model**




The following shows the classes in the above diagram and their attributes. Note that this meta-model has inheritance. For example, *DemandServiceRequest* inherits the attributes from *ServiceRequest*, so in addition to ServiceDemand, TimeUnits, and NumberOfVisits it also has the inherited attributes WorkloadName and ServerID.

| | | | |
|---|---|---|---|
| **Arc** | **Node** | **Server** | **Transit** |
| Description | Name | Quantity | To |
| FromNode | **Non-ServerNode** | SchedulingPolicy | Probability |
| ToNode | **OpenWorkload** | **ServiceRequest** | **Workload** |
| **ClosedWorkload** | ArrivalRate | WorkloadName | WorkloadName |
| NumberOfJobs | ArrivesAt | ServerID | **WorkUnitServer** |
| ThinkDevice | DepartsAt | **SinkNode** | TimeUnits |
| ThinkTime | TimeUnits | **SourceNode** | ServiceTime |
| TimeUnits | **QueueingNetworkModel** | **TimeServiceRequest** | **WorkUnitServiceRequest** |
| **DemandServiceRequest** | Name | NumberOfVisits | NumberOfVisits |
| NumberOfVisits | Date-Time | ServiceTime | |
| ServiceDemand | Description | TimeUnits | |
| TimeUnits | | | |

A *ServiceRequest* specifies the average *TimeService*, *DemandService* or *WorkUnitService* for each *Workload* that visits the *Server*. A *TimeServiceRequest* specifies the average service time and number of visits. A *Demand-ServiceRequest* specifies the average service demand (service time *x* number of visits). A *WorkUnitService-Request* specifies the average number of visits requested by each *Workload* that visits a *WorkUnitServer*. Upon completion of the *ServiceRequest*, the *Workload Transits* to other *Nodes* with a specified probability.

Changes to the PMIF 1.0 meta-model described in the next section were made because of differences in XML and CDIF. The enhancements described after that facilitate the exchange of models between unlike tools.

## 3.2. XML changes

Our approach was to use the PMIF 1.0 meta-model and develop an XML schema [13] that matched it as closely as possible. Classes in the meta-model became elements in the schema. Attributes in the meta-model became attributes in the schema.

XML allows inheritance, but its specification in XML is inconvenient. For example, consider a *Node* that is a *Server* and a *WorkUnitServer*. With inheritance it would be specified:

```
<Node Name="Sample">
  <Server Quantity="1" SchedulingPolicy="FCFS">
    <WorkUnitServer ServiceTime="0.05"
      TimeUnits="sec"/>
  </Server>
</Node>
```

By collapsing the inheritance hierarchy it turns into:

```
<WorkUnitServer Name="Sample" Quantity="1"
      SchedulingPolicy="FCFS" TimeUnits="sec"
      ServiceTime="0.05"/>
```

The latter is more readable, easier to generate and import. We preserve the inheritance in the meta-model diagram because we did not convert the meta-model to XML, but rather use XML to implement the transfer format. The meta-model is easier to comprehend with the inheritance.

The relationships in the PMIF 1.0 meta-model (e.g., RepresentsArrivalsFor) are not supported in XML schemas. Relationships must be converted to attributes,

elements, or dropped. PMIF 1.0 has 5 relationships. In PMIF 2.0 the association classes were preserved (e.g., *ServiceRequest*), the others became attributes and are no longer depicted on the meta-model.

### 3.3. Enhancements

These are necessary enhancements to the PMIF to adequately describe QNM between tools that have unlike model descriptions, such as *SPE·ED* and Qnap, and are not due to the XML implementation.

**3.3.1. Routing probabilities.** The PMIF 1.0 meta-model in [10] uses number of visits instead of routing probabilities, assuming that from the number of visits, and with the knowledge of the queueing network topology, routing probabilities can be calculated. This assumption is true for many of the queueing networks that model computer systems. However, it is not true for the general case. Based on the equations that relate number of visits to routing probabilities (shown below), there may be more unknowns than equations to solve for them when the unknowns are the routing probabilities. In most modeling cases, the knowledge of the network topology helps to reduce the number of unknowns up to a point where the number of equations is enough to calculate the probabilities. However, network topology and number of visits to each node are not (always) enough to calculate routing probabilities. Since some tools (for instance Qnap) use probabilities to specify a queueing network and to avoid losing the generality of the PMIF specification, routing probabilities are used instead of visits. The number of visits can always be calculated from the routing probabilities as follows [1]:

For open networks:

$$v_i = p_{0i} + \sum_{j=1}^{N} v_j p_{ji} \qquad (i = 1,..., N),$$

Where $p_{ji}$ is the routing probability from node $i$ to $j$, and $v_i$ is the mean number of visits of a job to the $i^{th}$ node. In open networks, the node with index 0 represents the external world.

The probabilities $p_{0i}$ are obtained from the external arrival rates, since $\lambda_{0i} = \lambda \cdot p_{0i}$, where $\lambda$ is the overall arrival rate from outside to an open network and $\lambda_{0i}$ is the arrival rate of jobs from outside to the $i^{th}$ node.

And for closed networks:

$$v_i = \sum_{j=1}^{N} v_j p_{ji} \qquad (i = 1,..., N).$$

Since there are only (N – 1) independent equations for the visit ratios in closed models, the $v_i$ can only be determined up to a multiplicative constant, and $v_1 = 1$ is usually assumed (the node with index 1 frequently represents the think device).

We added the routing probability specification as a *Transit* element with attributes To and Probability. The *Transit* element(s) are appended to the *ServiceRequest* element as in the following example:

```
<WorkUnitServiceRequest WorkloadName="Withdrawal"
        ServerID="DEV1" NumberOfVisits="8">
  <Transit To="CPU" Probability="1"/>
</WorkUnitServiceRequest>
```

A *ServiceRequest* has one or more *Transit* elements.

Even though probability specifications are adequate, we left visits in the meta-model and made them optional. Tools that have analytic solutions use them, and it follows the "import-friendly" strategy described later.

The probability specifications also make the *Arc* specifications redundant. We left them in PMIF 2.0, primarily because they provide an import-friendly way of specifying information for a diagram of a QNM. PMIF 2.0 does not contain information, such as coordinates, for drawing the diagram, but it will likely be a future extension.

**3.3.2. Workload entry.** It is also necessary to append *Transit* element(s) to specify the probability and where the *OpenWorkload* and *ClosedWorkload* go when they enter the system.

**3.3.3. BranchPoints.** In PMIF 1.0, a *BranchPoint* is "a convenient way to specify the origin or destination of multiple arcs." PMIF 2.0 eliminates them (see [12] for the rationale).

### 3.4. Other meta-model modifications

PMIF 1.0 specified a maximum length for names; we removed this restriction. If the importing tool has a limitation, it must adapt names accordingly. Section 6.4 discusses how this was handled for Qnap.

There are requirements for names that are used as IDs in XML that are not explicit in the meta-model. The formal definition of an XML name that is used as an ID is in http://www.w3.org/TR/2004/REC-xml-20040204/#id.

CDIF has a MetaIdentifier--a unique name that associates content in the transfer format to its formal definition; we deleted the it. The MetaIdentifier could be restored if PMIF were ever to become a standard whose content was carefully controlled.

PMIF 1.0 contained both a Name and an ID attribute. The ID served as a cross reference for other specifications. For example, in PMIF 1.0 two nodes, such as CPU and Disk, each have an (integer) ID, such as 1 and 2 respectively, then the Arc specification was:

```
(Arc QNM001.1 (FromNode #d1) (ToNode #d2))
```

where QNM001.1 is the MetaIdentifier. In PMIF 2.0, the exporting tool specifies the name of the nodes as an ID:

```
<Arc FromNode="CPU" ToNode="DISK"/>
```

Technically it shouldn't matter because the interchange format is machine processed. In practice, however, one needs to read the XML for testing, to determine what model is in the file, to investigate model results, etc.

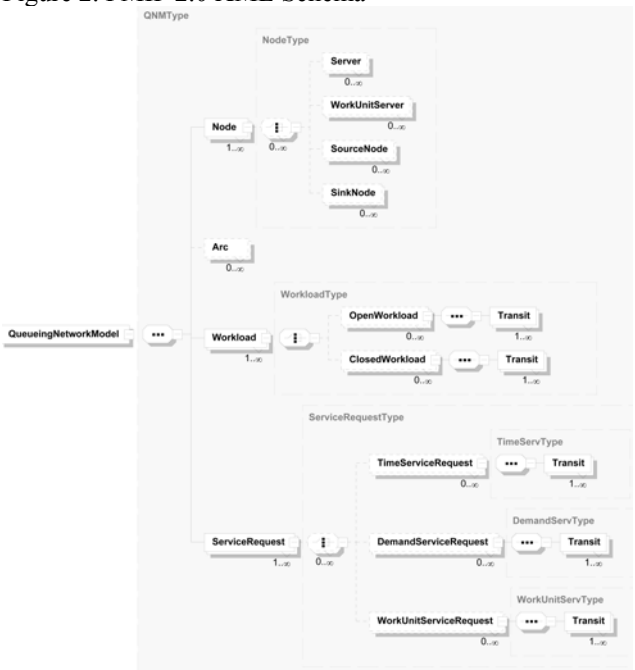
Other minor changes include:

- We added the NumberOfVisits attribute to the *DemandServiceRequest* to be import-friendly to tools that must specify the service time per visit rather than total service demand. While it is possible for the importing tool to calculate the visits, it can't currently be done with the simpler XSLT translation – it would require custom code.
- We deleted NodeType attribute on *Nodes* (redundant).
- We added optional attributes to *QueueingNetwork-Model* to specify date and time for the model. It is useful documentation, but we do not require it.
- We did not provide default values in the schema. Import-friendly tools will specify their own default values for optional attributes.

## 4. PMIF XML schema

The diagram of the XML Schema for PMIF 2.0 is in Figure 2. This schema definition requires that the elements be specified in the top-to-bottom order. *Nodes*, then *Arcs*, then *Workloads*, then *ServiceRequests*. We could not find a way to relax this ordering requirement (at the top level) in XMLSchema 1.1.

Figure 2. PMIF 2.0 XML Schema




There are several differences between Figure 2 and the meta-model in Figure 1. The *ServiceRequest* appears to be higher in the model hierarchy. This is because *ServiceRequest* is an association class in the meta-model, and when the inheritance hierarchy is "flattened" it associates the *Workload* with the *Node* (not the *Server*) even though only *Server* nodes will have a *ServiceRequest*. While we could have attached it to the *Server* instead, this method is more convenient for many exporting tools. For example, *SPE·ED* first exports the topology (*Nodes* and *Arcs*), then gets *Workload* information which contains intensity as well as service demands.

The following excerpt shows the schema definition for *Workload*. A *Workload* may be zero or more *OpenWorkloads* followed by zero or more *ClosedWorkloads*. *OpenWorkloads* have five attributes. They also have one or more associative *Transit* elements.

```
<xsd:complexType name="WorkloadType">
  <xsd:choice>
    <xsd:element name="OpenWorkload" minOccurs="0"
          maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Transit" type="TransitType"
          maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="WorkloadName" type="xsd:ID"
          use="required"/>
        <xsd:attribute name="ArrivalRate"
          type="nonNegativeFloat" use="required"/>
        <xsd:attribute name="TimeUnits"
          type="TimeUnitsType" use="optional"/>
        <xsd:attribute name="ArrivesAt" type="xsd:IDREF"
          use="required"/>
        <xsd:attribute name="DepartsAt" type="xsd:IDREF"
          use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ClosedWorkload" minOccurs="0"
          maxOccurs="unbounded">
      <! - -details omitted to save space, see [12]. - ->
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```

The following is a pmif.xml workload specification:

```
<Workload>
  <OpenWorkload WorkloadName="Withdrawal"
       ArrivalRate="1.0" TimeUnits="sec" ArrivesAt="Init"
       DepartsAt="Fini">
    <Transit To="CPU" Probability="1"/>
  </OpenWorkload>
  <OpenWorkload WorkloadName="Get_Balance"
       ArrivalRate="1.0" TimeUnits="sec" ArrivesAt="Init"
       DepartsAt="Fini">
    <Transit To="CPU" Probability="1"/>
  </OpenWorkload>
</Workload>
```

The complete schema definition may be seen at http://www.perfeng.com/pmif/pmifschema.xsd.

## 5. Import and export models using PMIF

The CDIF strategy is "export everything you know" and provide defaults for other required information; "import the parts you need and make assumptions if you require data not in the meta-model." Everything you know is not necessarily everything you use. For example, *SPE·ED uses* visits to specify routing, but it *knows* about probabilities, and it is relatively easy to calculate them. We created an "import-friendly" PMIF; that is, we include both visits and probabilities to make it easy on the import side. It is easy to do on output and it lets many importers use simple tools like XSLT rather than requiring custom code to do the import. Other import-friendly specifications are described in the following sections.

It is easy to check XML against a schema to confirm syntactic validity. That is, it contains everything it is supposed to, that IDREFS point to a declared ID, etc. This is useful for testing, and it is a good idea to validate a file before importing it. It is possible to add to the schema to validate model semantics. For example, that declared nodes are actually used in the model, the *TransitTo* attribute matches some *Arc*, etc. We did not include these things because it is reasonable to assume that production tools generate correct pmif.xml, and that it is only necessary to validate the semantics occasionally. We envision an *independent* tool that one could invoke as needed to validate the semantics of a QNM. It could do more than the simple checks mentioned above (e.g., confirm that *ClosedWorkloads* have a valid routing from the ThinkDevice through the system and back). That tool would be interesting further work.

Note that there is nothing in PMIF 2.0 or the XML schema that requires that *Transit* probabilities sum to 1. It was not a problem for Qnap. Other importing tools may need to handle the possibility that they do not sum to 1.

The following sections discuss specific issues in exporting from *SPE·ED* and importing into Qnap. Later we also discuss issues in the reverse exchange although this exchange was not implemented as part of this project.

### 5.1. Exporting a *SPE·ED* model into pmif.xml

*SPE·ED* uses the Document Object Model (DOM) [13] to export the pmif.xml. It creates the entire document in memory, then writes it to a file. Elements and attributes can be added in any order as long as they are in the correct location. It is a relatively small file, e.g., 2-3K for these examples, so the memory requirements are modest.

*SPE·ED* uses a standard topology for models. Each facility contains a CPU and one or more other types of devices. Within a facility the QNM is assumed to be a central server model. A model may contain multiple facilities, each with this central server topology.

*SPE·ED* does not have explicit source, sink, or think nodes. They had to be created for the pmif.xml along with their *Arcs*, and the *Transit* probabilities for the *OpenWorkload* and *ClosedWorkload*. *SPE·ED* also uses visits rather than probabilities. Probabilities were calculated for the *ServiceRequests*.

*SPE·ED* has the ability to specify a Quantity for each type of device. For the CPU, the quantity is the number of multi-servers fed from a single queue, so it suffices to specify the Quantity for the CPU Server in the pmif.xml. Other devices, such as disks, also have a quantity, however each of those devices has a separate queue. *SPE·ED* assumes that the visits are equally spread to those devices. To export devices with a quantity greater than one, it is necessary to generate separate servers, separate arcs, and calculate equal probabilities for the *Transit* elements from the CPU. Several tools including Qnap have the capability to represent arrays or lists of servers. We decided not to add a capability to PMIF 2.0 to accommodate them because it would put a burden on importing to tools that do not have this capability.

Finally, *SPE·ED* has no restriction on special characters in names. We transform characters that are not allowed in XML IDs into an underscore.

### 5.2. Importing a pmif.xml model into Qnap

Qnap reads the input (QNM specification and solving parameters) from a file. Ultimately, Qnap would have an interface that would read from its standard file OR the pmif.xml file. However, we did not have access to 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.

We generated an XSLT specification that transforms a pmif.xml file into a file that is read and executed by Qnap. The direct use of XSLT was feasible due to the possibility of specifying the stations by parts in the Qnap input file. This might not be possible for other tools with stricter ordering in the input file, in which case two possibilities would arise: The use of DOM to import pmif.xml, or the use of XSLT together with a conventional programming language. The use of XSLT is fairly simple; therefore we recommend XSLT when possible for the translation into a tool's file format.

For an implementation that reads from the XML file directly, the use of DOM would be necessary since XSLT can only transform an XML file into another file. It would be advisable to read the entire pmif.xml file into memory then interpret and insert parameters into appropriate internal data structures because of the ordering in the

XML schema. That is, some transformations may require information from elements that have not been read yet.

The translation from pmif.xml to Qnap, required several special considerations. The most important ones are discussed in the following paragraphs. Others are in [12].

Qnap needs a separate source node for each open workload; therefore those had to be generated. On the other hand, Qnap does not have an explicit sink node. The specification for a client leaving the system is in the transit using a specific identifier (OUT). Hence, this situation had to be detected and the specific *Transit* generated.

Qnap needs service time rather than total demand. We put visits on the *DemandServiceRequest* to make the prototype translation easier. Ultimately we need a set of conversion routines in a developer kit to handle probability/visit computations.

Time units between *SPE·ED* and Qnap are not a problem. *SPE·ED* uses seconds for all specifications. Qnap just needs consistent time units. Ultimately we need time conversion routines in a developer tool kit.

In Qnap only the first 8 characters of an identifier are significant. The prototype truncates the names, but we do not test that the names are unique. Ultimately we need a routine that does this check and generates unique names if necessary. Other types of checking are also required for the identifiers (see [12] again).

Qnap allows the models to be solved analytically or through simulation and this needs to be indicated in the Qnap input file. Therefore, we implemented two different transformations, one for each of those solutions. The model specification is the same, the only difference being in a few parameters related to the execution.

Solving instructions and instructions for generating results are also specified in the Qnap input file. These include some specific instructions for generating extra results in order to be able to compare results given by both tools (*SPE·ED* and Qnap). There are also differences in stopping conditions in the two tools that makes the comparison of simulation results approximate.

### 5.3. Exporting a Qnap model into pmif.xml

Even though the exportation of a Qnap model into pmif.xml is out of the scope of our prototype, we point out a few issues that would arise when trying to do so. One possibility is to generate a pmif.xml from a Qnap input file. Another is to change Qnap code to generate pmif.xml from the internal QNM information. Since we did not have access to the Qnap source code, below are the main issues for the first option. Some of these points would also apply with the second option since they come from the "way of thinking" in Qnap.

- Qnap has many default values, for example Quantity=1, SchedulingPolicy=FCFS. If those attributes are not defined they have to be created for the pmif.xml.
- It is not explicitly said whether a Qnap class is open or closed. It has to be detected from the collection of transit definitions for each workload. Another possibility would be to ask the user about it.
- If the model has only one workload, there is no need of having any workload definition in Qnap. Therefore this situation needs to be detected and a workload created.
- Qnap does not have an explicit sink node, it must be created when the model contains open workloads.
- Separate servers must be generated for devices that are manipulated as a list in Qnap.
- Some characters are allowed in a Qnap identifier but not in an ID in XML.

### 5.4. Importing a pmif.xml model into *SPE·ED*

Importing pmif.xml into *SPE·ED* would require a DOM interface for the following reasons.

*SPE·ED* is a software performance modeling tool. It uses QNMs for system execution models. So importing a pmif.xml only specifies the system execution model. *SPE·ED* would create a scenario for each workload defined in pmif.xml, but the scenario would only have one processing step because PMIF 2.0 does not represent the software execution structure. To import software processing details one would use an SPE meta-model such as that defined in [14] rather than a PMIF.

*SPE·ED* also specifies software resource requirements (e.g., database accesses) rather than computer resource requirements (e.g., CPU and disk). PMIF 2.0 specifies service requirements for each device. So it would be necessary to convert *ServiceRequest* specifications into appropriate internal structures in *SPE·ED*.

*SPE·ED* has a concept of *facilities* as collections of central server models. It would require an analysis of the topology of models with multiple facilities to identify the devices that go with facilities. Hints, like adding an optional FacilityId to node specifications, or asking the user to identify facilities, would be helpful.

*SPE·ED* has a limit on the number of distinct devices in each facility. A "quantity" specification allows more actual devices. So a straightforward translation of devices may exceed the allowable number, and it would be necessary to determine which devices could be combined with the Quantity specification, and to handle excess devices.

*SPE·ED* also has special network devices that connect multiple facilities. It would be difficult to detect these in a general pmif.xml file, so it would be necessary to query the user to determine if those devices are present.

The (optional) NumberOfVisits is useful for *SPE·ED*, so an import-friendly sending tool should fill them in.

## 6. Prototype implementation and results

### 6.1. Prototype: *SPE·ED* to pmif.xml

The *SPE·ED* prototype implemented custom code using the DOM to create the pmif.xml for user's model (project). The following is an excerpt of the C++ code to create the *OpenWorkload* element in the pmif.xml:

```
elementName = ::SysAllocString(L"OpenWorkload");
attr2Name = ::SysAllocString(L"ArrivalRate");
attr3Name = ::SysAllocString(L"ArrivesAt");
attr4Name = ::SysAllocString(L"TimeUnits");
attr5Name = ::SysAllocString(L"DepartsAt");
element = CreateDOMNode(pDoc,
      MSXML::NODE_ELEMENT, elementName);
SetStringAttribute(element, nameAttr, thename);
SetFloatAttribute(element, attr2Name, in_low);
SetStringAttribute(element, attr3Name, "SourceNode");
SetStringAttribute(element, attr5Name, "SinkNode");
SetStringAttribute(element, attr4Name, "sec");
CHECKHR(pParent->insertBefore(element, after, &p1));
SAFERELEASE(element);
addTransit(p1, cpuName, 1.0);
SAFERELEASE(p1);
```

An earlier example showed the resulting pmif.xml for an *OpenWorkload*. The complete pmif.xml file for the ATM example described in [10] is in [12].

### 6.2. Prototype: pmif.xml to Qnap

The following is an excerpt of the XSLT code to transform the pmif.xml to Qnap format. In Qnap a server needs first to be declared (a), and after all the declarations, the stations can be specified (b).

(a)

```
 / DECLARE/ QUEUE <xsl:for-each
select="QueueingNetworkModel/Node/Server">
    <xsl:if test="position() != 1">, </xsl:if>
    <xsl:value-of select="substring(@Name,1,8)"/>
</xsl:for-each>;
```

(b)

```
<xsl:for-each select="QueueingNetworkModel/Node/Server">
/STATION/ NAME=<xsl:value-of
          select="substring(@Name,1,8)"/>;
  <xsl:choose>
    <xsl:when test='@SchedulingPolicy = "IS"'>
      TYPE = INFINITE;
    </xsl:when>
    <xsl:when test="@Quantity > 1">
     TYPE = MULTIPLE(<xsl:value-of select="@Quantity"/>);
    </xsl:when>
  </xsl:choose>
  <xsl:choose>
    <xsl:when test='@SchedulingPolicy = "PS"'>
      SCHED = PS;
      </xsl:when>
    <xsl:when test='@SchedulingPolicy = "FCFS"'>
      SCHED = FIFO;
      </xsl:when>
  </xsl:choose>
</xsl:for-each>
```

### 6.3. Prototype validation

We conducted tests on the ATM model from [10] and most of the models covered in [11]. The excluded book models had major performance problems and their system execution model had saturated devices and was thus unstable. Early tests confirmed that the transferred model was unstable, and we eliminated those models.

Table 1 shows a subset of the models with the solution method (simulation or analytic) in parenthesis along with the response time, CPU utilization, Disk utilization and confidence and simulated time from each tool. See [12] yet again for complete results. Note that the *SPE·ED* run time is the same as Qnap, however it is reported as less because of a reset after a start interval. There are also differences in the way the confidence is handled. *SPE·ED* uses a (user-selected) confidence interval for the overall response time, and uses a default 70% confidence level. This is because it is intended to simulate software architectures and designs at a time when exact resource specifications are imprecise. So there is no point in simulating for a long period of time to get precision in the solution when the model parameters are only approximate. Qnap, however, uses a 95% confidence level and reports the interval by device rather than overall. It is intended to maximize the precision of the solution.

The first 2 sets of results are for the ATM example from [10]. It is an open model with 2 workload classes. Note that *SPE·ED* does not solve multi-class models analytically; the analytic results from Qnap are for comparison to the simulation results. This example shows that allowing comparison of multiple solution techniques across tools is a valuable benefit of the PMIF, and it confirms that the transfer was successful.

Model studies 3 and 4 are the Drawmod example Architecture 3 from [11]. It is a single class, closed model. The results are for 10 users with a 60 sec. think time.

Model studies 5 and 6 are for the revised version of the telephone switching example (POTS) in [11]. It is a multiclass open model. Study 5 shows the analytical results from Qnap. There are no disks in this model, the disk column instead reports results for the Line Interface.

The results confirm that the pmif.xml successfully transfers models between the two tools. The comparison of solutions led to the discovery and correction of an inconsistency of *SPE·ED* analytical and simulation solutions. When analytical and simulation solutions differ slightly, it is difficult to determine whether the difference is statistically significant or if it results from an error in the solver(s). The ability to easily compare solutions across tools is valuable both for tool developers and users.

**Table 1. Model Results for Example Files**

| Model Study | Response Time | | CPU Utilization | | Disk Utilization | | Confidence / SimTime | |
|---|---|---|---|---|---|---|---|---|
| | ***SPE·ED*** | **Qnap** | ***SPE·ED*** | **Qnap** | ***SPE·ED*** | **Qnap** | ***SPE·ED*** | **Qnap** |
| 1. ATM (S) | | | | | | | .314/ 49890 | 95% / 50000 |
| Withdrawal | 11.971 | 11.9 | 0.006 | 0.0063 | 0.403 | 0.3984 | | |
| GetBalance | 6.354 | 6.362 | 0.003 | 0.0025 | 0.151 | 0.1519 | | |
| 2. ATM (A) | | | | | | | | |
| Withdrawal | | 11.9 | | 0.0063 | | 0.4 | | |
| GetBalance | | 6.336 | | 0.0025 | | 0.15 | | |
| 3. Drawmod3.2 (A) | 12.55 | 12.49 | 0 | 0.00049 | 0.45 | 0.447 | | |
| 4. Drawmod3.2 (S) | 12.6 | 12.7 | 0 | 0.00049 | 0.45 | 0.4465 | .29/ 49585 | 95% / 50000 |
| 5. POTS2 (A) | | | | | | **LineIF** | | |
| CallOrigination | | 0.4773 | | 0.369 | | 0.135 | | |
| CallTermination | | 0.2883 | | 0.221 | | 0.09 | | |
| HangUpCalled | | 0.063 | | 0.05 | | 0.015 | | |
| HangUpCaller | | 0.0996 | | 0.077 | | 0.03 | | |
| 6. POTS2 (S) | | | | | **LineIF** | **LineIF** | .370 / 49900 | 95% / 50000 |
| CallOrigination | 0.3697 | 0.4833 | 0.323 | 0.3694 | 0.135 | 0.135 | | |
| CallTermination | 0.2546 | 0.2913 | 0.221 | 0.2204 | 0.09 | 0.090 | | |
| HangUpCalled | 0.0633 | 0.063 | 0.05 | 0.0493 | 0.015 | 0.015 | | |
| HangUpCaller | 0.0979 | 0.1002 | 0.076 | 0.0768 | 0.03 | 0.030 | | |

## 7. PMIF 2.0 extensions

XML does not provide the same concept as CDIF *levels* for specifying extensions. It is possible, however, to define different versions of the schema. Versions could be used to add additional features not present in PMIF 2.0.

Some features that are relatively easy to add include:

- Additional SchedulingTypes, e.g., LIFO, Quantum
- Priorities for Workloads along with priority scheduling and preemption
- Service time distributions, e.g., EXP, HEXP, CST, Erlang, Uniform, Cox

PMIF 2.0 assumes service distributions that permit exact, analytic solutions. They are not explicit in the meta-model.

Several other features are available in Qnap and other similar tools, such as maximum queue capacity, semaphores (events), workload phases and phase changes, mailboxes, passive resource queues, and other advanced features, would be useful in a future version of PMIF.

Reviewers of earlier versions of PMIF suggested that it should contain specifications for solving the model and for the results that should be produced. PMIF 2.0 does not yet include them. We also used default stopping conditions for the simulation runs. A run length specification seems universal, but it may be difficult to find more advanced conditions common to diverse tools. For example, some use confidence levels others confidence intervals, and the method used varies (e.g., batch means with varying batch sizes, spectral method, etc.).

Furthermore, Qnap and other tools allow the user to specify the solution method (such as, convolution, MVA, normalized convolution, iterative approximation, etc.). So a specification for the solution method should be included as an optional attribute for model solution specifications.

The PMIF meta-model (and schema) could specify the results, or a separate one could be defined for them. A comma separated text file of those results would be convenient for importing into spreadsheets, databases or other tools. We prefer to customize results to the problem rather than producing all possible results because too many are almost as bad as too few.

The choices for each of these extensions depend on how one wants to use the PMIF. The requirements for solving the model and producing results differ in the six cases listed in Section 1. Some need an automatic solution and results, while others only need to get the model into another tool and the user can then fill in missing details. Therefore, we may want several versions of the PMIF for different purposes, or we may want to have separate schemas, produce multiple files (e.g., model, solution, and results), and merge them as necessary. Further study is required to determine what options should be provided and how to represent them in the meta-model(s).

## 8. Status & future work

The initial prototypes for exporting *SPE·ED* models and importing into Qnap are complete. *SPE·ED* currently exports models with multiple workloads, but they must all

execute on the same facility. Multiple facilities will be added in the near future. They were omitted to focus on the essence of the interchange problems.

Section 5 described several of the features omitted from the initial Qnap XSLT prototype because they were not problems in the models studied. They need to be completed for a production version of the interchange.

We would like to create a pmif.xml developer kit that would help others implement an export and import capability for additional tools. It would contain the information in this paper, some additional advice on tools and information learned in this project, a semantic validator for testing, sample models and results, standard subroutines for name conversions, visit/probability calculations, and other common functions.

## 9. Conclusions

The PMIF supplies users and tools developers with an exchanging mechanism of system model information based on the queueing networks formalism. The exporting and importing tools can either support the PMIF or provide an interface to read/write model specifications from/to a PMIF file. We have presented a new version of the PMIF (PMIF 2.0) and its XML implementation. We have also proved the concept with the development of a prototype in which the exporting tool is *SPE·ED* and the importing tool is Qnap. Different types of models have been used as transfer examples and results prove that the exchange is successful. Moreover, it has been shown that the comparison of multiple solution techniques across tools can lead to a wide range of benefits.

We originally viewed the lack of access to Qnap source code as a serious limitation on the project. It turns out to be a significant result that pmif.xml can be used by anyone to exchange information relatively easily between two tools that provide a file input/output capability. It does not require the tool developer to modify code to be of use. Of course, it would be nice if tool developers would support it so that users would not have to go to this extra effort. We propose that those who do develop XSLT or custom code routines to go between pmif.xml and file interfaces make it available to others to promote the easy interchange of models. The PMIF 2.0 XML schema is available at http://www.perfeng.com/pmif/pmifschema.xsd.

## 10. Acknowledgements

The authors would like to thank Ramon Puigjaner and the ACSIC research group at the *Universitat de les Illes Balears* for the help offered towards their collaboration.

## 11. References

1. Bolch, G., et al., *Queueing Networks and Markov Chains. Modeling and Performance Evaluation with Computer Science Applications*. 1998: Wiley Interscience.
2. Cortellessa, V., *From UML to Execution Graphs and Queueing Networks: Design and Implementation of the XML-based tool XPRIMAT*. 2004.
3. Cortellessa, V. and R. Mirandola. *Deriving a Queueing Network based Performance Model from UML Diagrams*. in *Proc. Workshop on Software and Performance*. 2000. Ottawa: ACM.
4. EIA, *CDIF - CASE Data Interchange Format Overview*, Engineering Department, Electronics Industries Association, Arlington, VA, EIA/IS-106, January 1994.
5. Gu, G. and D. Petriu. *XSLT Transformation from UML Models to LQN Performance Models*. in *Proc. Workshop on Software and Performance*. 2002. Rome: ACM.
6. L&S, *Computer Technology, Inc., Performance Engineering Services Division*, in *# 110, PO Box 9802*, (505) 988-3811, www.perfeng.com: Austin, TX 78766.
7. Simulog, Paris *Office*, in *1 rue James Joule, 78286 Guyancourt Cedex*, 33 (0)1 30 12 27 00, www.simulog.fr:
8. Smith, C.U. and L.G. Williams. *Panel Presentation: A Performance Model Interchange Format*. in *International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1995. Heidelberg Germany: Universitat Dortmund, Germany.
9. Smith, C.U. and L.G. Williams, *Definition of a Performance Model Interchange Format*, L&S and SER, www.perfeng.com publications link, February 1995.
10. Smith, C.U. and L.G. Williams, *A Performance Model Interchange Format*. Journal of Systems and Software, 1999. **49**(1).
11. Smith, C.U. and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. 2001: Addison-Wesley.
12. Smith, C.U. and C.M. Lladó, *Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation Technical Report*, L&S Computer Technology, Inc., www.perfeng.com/paperndx.htm, April 2004.
13. W3C, *World Wide Web Consortium*, www.w3c.org, 2001.
14. Williams, L.G. and C.U. Smith. *Information Requirements for Software Performance Engineering*. in *Proceedings 1995 Int. Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1995. Heidelberg, Germany: Springer.
15. Wu, X. and C.M. Woodside. *Performance Modeling from Software Components*. in *Proc. Workshop on Software and Performance*. 2004. Redwood Shores, CA: ACM 488043.