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

Connie U. Smith† and Catalina M. Lladó §
†Performance Engineering Services,
PO Box 2640, Santa Fe, New Mexico, 87504-2640 USA
(505) 988-3811

§ Universitat Illes Balears
Departament de Matemàtiques I Informática
Carretera de Valldemossa, Km. 7.6, 07071 Palma de Mallorca, Spain

February 2004

Copyright © 2004, Performance Engineering Services
All rights reserved
This material may not be sold, reproduced or distributed without written permission from
Performance Engineering Services
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 conduct several studies, then move the model to a tool better suited to network analysis. Other uses for the PMIF include:

1. A user or tool developer may want to compare solutions from multiple tools
2. A user may want to create input specifications in pmif.xml or in a familiar tool rather than learn the interface to multiple tools
3. A user may want to migrate a model to temporarily use another tool to develop and study more detailed models
4. A user may want to migrate a model to “permanently” use a different tool for the model.
5. A user may want to create software performance models to study architecture and design trade-offs, then use another tool to study the computer system operating environment in greater detail.
6. A user may want to compare different tools, for instance 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 ground that all tools may use as an interface. 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 4· (N! / (2!(N-2)!)) requirement for customized interfaces. With PMIF, tools export and import with the same format so the requirement for customized interfaces is reduced to 2·N. With XML tools the complexity and amount of effort to create the PMIF interface is quite small [W3C 2001]. 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 paradigm that calls for defining the information requirements for a Queueing Network Model (QNM) with a meta-model [Smith and Williams 1995b; 1999]. 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 then implemented a prototype export mechanism from the \textit{SPE\textregistered\textsc{ed}} software performance modeling tool \cite{L&S} into pmif.xml, and a companion prototype import mechanism from pmif.xml into the Qnap system performance modeling tool\footnote{Qnap is a modeling tool that can be used on its own or through the use of 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.} \cite{Simulog}. 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 with other tools.

There has been some 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 \cite{Cortellessa and Mirandola 2000}. More recently, Coretellessa, 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 \cite{Coretellessa 2004}. Gu and Petriu use XSLT (eXtensible Stylesheet Language for Transformations) \cite{W3C 2001} to transform UML models in XML format to the corresponding Layered Queueing Network (LQN) description which can be read directly by existing LQN solvers \cite{Gu and Petriu 2002}. Wu and Woodside use an XML Schema to describe the contents and datatypes that a Component-Based Modeling language (CBML) document may have \cite{Wu and Woodside 2004}. CBML is an extended version of the Layered Queueing Network (LQN) language that adds the capability and flexibility 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 of the PMIF transfer format and the modifications that were required. Section 4 describes the resulting XML schema for PMIF 2.0. Section 5 covers the examples that we studied. Then the rest of the paper reports on the status, describes some extensions and future work, and conclusions. Appendices contain a brief primer on the XML features; the complete schema; the pmif.xml for the ATM example; the XSLT for the translation from pmif.xml to Qnap; the resulting Qnap input and output files for the ATM example; and some experiences and lessons learned building these tools.

\section{PMIF Summary}

\subsection{Paradigm}

The approach selected for PMIF 1.0 was based on the EIA/CDIF (Electronic Industries Association/CASE Data Interchange Format) standard \cite{EIA 1994}. CDIF is a family of standards that describe a mechanism for transferring information between CASE tools. The standards define a transfer format that allows tools that have different internal databases and storage formats to exchange information. An exchange takes place via a file and internal tool information is translated to and from the file’s transfer format.

In the CDIF standard, the information to be transferred between two tools is known as a \textit{model}. The contents of a model are defined using a \textit{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.”
To export models with CDIF, tools provide all data they have that is specified in the meta-model. Tools must provide default values for the essential data in the CDIF 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. 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.

To import models in the CDIF format, tools use the data provided, discard data items they do not need, and make assumptions about data items they require that are not in the basic meta-model. 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.

After importing models, the users of the tools will likely need to input 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.

The CDIF paradigm was used to define a QNM meta-model, and a case study illustrated its use [Smith and Williams 1999]. The transfer format used 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. So 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.

### 2.2 PMIF 1.0 Meta-Model Development

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 [Smith and Williams 1999]. A wide variety of features and terms were considered, as well as feedback from researchers in the performance field, to ensure that PMIF 1.0 adequately described the information requirements.

PMIF 1.0 was an initial investigation to establish the viability of the approach. It started with a meta-model of the information required for a manageable subset: the data needed for QNM that may be solved using exact analytic solution algorithms. It was envisioned that the CDIF concept of levels could be used to describe additional facets of QNM. In CDIF, Level 0 describes the fundamental subset of information required, Level 1 adds some additional features, Level 2 adds more, etc.

This work also begins with PMIF 1.0 as a starting point to establish the viability of using XML as the transfer format. XML does not provide the same concept of levels, but it does provide for versions of schemas. We will discuss this mechanism for extensions later. 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 and the modifications that were made and why.

---

2 CDIF was derived from the Electronic Design Interchange Format (EDIF) used to transfer VLSI designs among design tools. LISP was the language of choice at the time because many of those design tools ran on LISP workstations.
3 QNM Meta-Model 2.0 Description and Modifications

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. The first is to provide a rigorous definition for the information required for a QNM that may be solved using exact analytical techniques. The second purpose of the meta-model is to generate the formal PMIF using the XML transfer format derived from the meta-model.

The UML meta-model diagram is shown in Figure 1. This figure is based on the PMIF meta-model defined in [Smith and Williams 1999]. This version converts the diagram to UML, adds additional information to the meta-model and eliminates other information. The following paragraphs describe the classes and their attributes. Subsequent sections provide more details on the content changes in version 2.0.

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. A Server may be a WorkUnitServer that executes a fixed amount of work (processing service) for each Workload that makes a request for service.

- **Non-ServerNode** represents nodes that show topology of the model, but do not provide service. There are two types of Non-ServerNodes
  - **SourceNode**: represents the origin of an OpenWorkload.
  - **SinkNode** represents the exit point of an OpenWorkload.

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**: 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.

Upon arrival, a Workload Transits to other Nodes with a specified probability.

A service request associates the Workloads with Servers. A ServiceRequest specifies the average TimeService, DemandService or WorkUnitService 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. A WorkUnitServiceRequest 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.
3.2 Modifications to the PMIF 1.0 Metamodel

During this project, we made several changes to the PMIF meta-model to resolve some limitations and some difficulties in exchanging models between unlike tools. Changes described in the next section were made because of differences in XML and CDIF. The enhancements described after that were made to facilitate the exchange of models between unlike tools.
3.3 XML Changes

Our approach was to use the PMIF 1.0 meta-model and develop an XML schema [W3C 2001] 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:

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

By collapsing the inheritance hierarchy it turns into:

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

The latter is more readable and easier to generate and import. We preserve the inheritance in the meta-model diagram because we are not trying to 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. Table 1 shows how they are handled in PMIF 2.0.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Description</th>
<th>PMIF 1.0</th>
<th>PMIF 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsConnectedTo</td>
<td>Connects one node to another in a QNM</td>
<td>Associative entity Arc</td>
<td>Association class Arc</td>
</tr>
<tr>
<td>IsComposedOf</td>
<td>A QNM is composed of one or more Nodes, zero or more Arcs, and one or more Workloads</td>
<td>Defines required Entities</td>
<td>The ServiceRequest element(s) are added to the QNM composition when the hierarchy is flattened.</td>
</tr>
<tr>
<td>ProvidesServiceFor</td>
<td>Connects a Server to its provision of service for a Workload.</td>
<td>Associative entity ServiceRequest</td>
<td>Association class ServiceRequest</td>
</tr>
<tr>
<td>IsPairedWith</td>
<td>Connects the SourceNode and the SinkNode in the QNM</td>
<td>Explicit relationship</td>
<td>Attributes ArrivesAt and DepartsAt of OpenWorkload</td>
</tr>
<tr>
<td>RepresentsArrivalsFor</td>
<td>Connects the SourceNode and the OpenWorkload arrivals in the QNM</td>
<td>Explicit relationship</td>
<td>Attribute ArrivesAt of OpenWorkload</td>
</tr>
</tbody>
</table>

3.4 Enhancements

Some enhancements to PMIF were discovered during implementation because the two tools we used had unlike model descriptions. These are necessary enhancements to the PMIF to adequately describe QNM between tools and are not due to the XML implementation.

3.4.1 Routing Probabilities

The meta-model defined in [Smith and Williams 1999] proposes the use of 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 [Bolch et al. 1998]:

For open networks:

\[ v_i = p_{0i} + \sum_{j=1}^{N} v_j p_{ji} \quad (i = 1, \ldots, N), \]

Where the \( p_{ji} \) is the routing probability from node \( i \) to node \( j \) 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 to the network.

The probabilities \( p_{0i} \) are obtained immediately 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:

\[ v_i = \sum_{j=1}^{N} v_j p_{ji} \quad (i = 1, \ldots, N). \]

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.

We chose to add 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:

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

There are one or more Transit elements per ServiceRequest.

Even though probability specifications are adequate, we left visits in the meta-model and made them optional. They are used by tools that have analytic solutions, 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.

### 3.4.2 Open Workloads

With the addition of routing probabilities, it is also necessary to add Transit element(s) to specify the probability and where the OpenWorkload goes when it leaves the SourceNode. These Transit element(s) are appended to the OpenWorkload element as in the following example:

```xml
<OpenWorkload WorkloadName="Get_Balance" ArrivalRate="1.0" TimeUnits="sec" ArrivesAt="Init" DepartsAt="Fini">
  <Transit To="CPU" Probability="1"/>
</OpenWorkload>
```
### 3.4.3 Closed Workloads

PMIF 1.0 has a relationship to associate an *OpenWorkload* with its *SourceNode*. It also needs a relationship to associate a *ClosedWorkload* with a node that represents the think device. PMIF 2.0 adds an attribute to the *ClosedWorkload* for ThinkDevice.

With the addition of routing probabilities, it is also necessary to add a *Transit* element to specify where the *ClosedWorkload* goes when it leaves the ThinkDevice. These *Transit* element(s) are appended to the *ClosedWorkload* element as above.

### 3.4.4 BranchPoints

In PMIF 1.0, a *BranchPoint* is “a convenient way to specify the origin or destination of multiple arcs.” They are primarily useful for diagrams of QNM; they have no function in the model solution. PMIF 2.0 eliminates *BranchPoints* because they are problematic for tools that do not support them, such as Qnap. If *BranchPoints* were specified in the *Transit* elements, Qnap (and other such tools) would need to know the node(s) that would ultimately be reached as a result of the *Transit*. It is not possible to substitute a “dummy” node with no service requirement for the *Transit*. This transformation is not supported by XSLT, so the import would require custom coding. There are few tools that require *BranchPoints*, so the elimination should not be a problem. When diagram information extensions are developed, another method of representing multiple arc origins and destinations should be developed that will not affect the model solution.

### 3.5 Other Meta-model Modifications

PMIF 1.0 specified a maximum length of 32 characters for names. We removed this restriction from the XML schema. Most tools have a restriction on the length of names, but it does not seem reasonable to set a maximum length in the transfer format and thus limit tools that provide for longer names for readability. If the importing tool requires a shorter name, it must adapt names accordingly. Section 6.4 discusses how this was handled for importing names to Qnap.

There are requirements for names that are used as IDs in XML that are not explicit in the meta-model. The following is a formal definition of an XML name that is used as an ID:

\[
\text{ID is as NCNAME}
\]
\[
\text{NCName ::= (Letter | `\_`)\(\text{NCNameChar}\)*}
\]
\[
\text{NCNameChar ::= Letter | Digit | `\_` | `\.` | `-` | CombiningChar | Extender}
\]

You can see the last two in (http://www.w3.org/TR/2004/REC-xml-20040204/#NT-CombiningChar). We decided to avoid them.

CDIF has a MetaIdentifier – a unique name for each entity, attribute, and relationship that associates content in the transfer format to its formal definition. We included annotations in the schema with the definition of the elements and attributes, but did not see a reason to include the MetaIdentifier because the schema itself provides a mechanism for identifying XML content. The MetaIdentifier might be useful if PMIF were ever to become a standard whose content was carefully controlled. The MetaIdentifier could be re-added to the schema at that time.

PMIF 1.0 contained both a Name and an ID attribute. The ID served as a cross reference for other specifications. For example, an *Arc* specifies a FromNode and a ToNode, and used a (unique) ID to reference the nodes. In XML it is easy to declare that a name is an ID, and use the name as an IDREF for the FromNode and ToNode. For example, in PMIF 1.0 for two nodes, such as CPU and Disk, each would have an (integer) ID, such as 1 and 2 respectively, then the Arc specification was:

(Arc QNM001.1)
where QNM001.1 is the MetaIdentifier. In PMIF 2.0, the exporting tool specifies the name of the nodes rather than their ID as in the following example:

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

The latter is more readable. Technically it shouldn’t matter because the interchange format is supposed to be machine processed. In practice, however, one needs to read the XML occasionally for testing, to determine what model is in the file, to investigate model results, etc. This technique has the additional advantage that it is easy to do a validity check on the file to make sure that there is a declaration of Nodes that are referenced by Arcs. Validity checks are discussed later.

Other minor changes include:

- Some of the attributes, such as NumberOfVisits, became optional in PMIF 2.0 because they are no longer essential. We elected to specify them, though, to create an import-friendly interchange.
- 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.
- The NodeType attribute on Nodes is deleted because it is redundant. For example, it isn’t necessary to specify `<SourceNode NodeType="Source" .../>`
- We added optional attributes to the QueueingNetworkModel to specify a date and time for the model. It is our experience that this is useful documentation, but we do not require everyone to specify it or import it.

We did not provide default values in the schema. We could have specified some typical values, e.g., Quantity="1" and others. We have left it up to the exporting tool to specify a default value, or if it is an optional attribute that is not specified, the importing tools must make their own assumption. Import-friendly tools will specify their own default values for optional attributes.

We also found a few minor errors in the use of terms in earlier papers that described PMIF 1.0 [Smith and Williams 1995a; 1997; 1999]:

- The original meta-model definition and diagram used the term BranchPoint, but the example used the term BranchNode. PMIF 2.0 eliminates BranchPoints so it is not necessary to correct the inconsistency here.
- The original meta-model definition and diagram used the term QueueingNetworkModel, but the example used the term “QNM Model”. PMIF 2.0 uses QueueingNetworkModel.

4 PMIF XML Schema

The diagram of the XML Schema for PMIF 2.0 is in Figure 2.3 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.

---

3 This diagram was produced by XMLSpy from Altova (www.altova.com).
There are several differences between Figure 2 and the meta-model in Figure 1. The most obvious is that 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`. The `QueueingNetworkModel` definition requires one or more `Workloads`. A `Workload` may be zero or more `OpenWorkloads` followed by zero or more `ClosedWorkloads`. `OpenWorkloads` have `WorkloadName`, `ArrivalRate`, `TimeUnits`, `ArrivesAt`, and `DepartsAt` attributes. They also have one or more associative `Transit` elements that specify where the `Workload` goes when it leaves the `SourceNode` specified by `ArrivesAt`.

```xml
<xsd:complexType name="WorkloadType">
  <xsd:choice maxOccurs="unbounded">
    <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">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>
```
The following excerpt shows the pmif.xml workload specification based on this schema for the ATM Example in [Smith and Williams 1999]:

```xml
<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](http://www.perfeng.com/pmif/pmifschema.xsd).

5 Import and Export Models using PMIF

5.1 Philosophy

The CDIF strategy is “export everything you know” and “import the parts you need.” 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. The redundant specifications are currently optional. Other import-friendly specifications are described in the following sections.

It is easy to check XML against a schema to confirm that it is syntactically valid. That is, it contains everything it is supposed to, in the proper location, 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 would have been possible to add some things to the schema to further confirm that the semantics of the model are correct. For example, that declared nodes are actually used in the model, the Transit To matches some Arc, etc. We decided not to add these things to the PMIF schema. 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. For example, it could confirm that ClosedWorkloads have a valid routing from the ThinkDevice through the system and back to the ThinkDevice. 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, from Qnap into SPE·ED, although this exchange was not implemented as part of this project.

5.2 Exporting a SPE·ED model into pmif.xml

SPE·ED uses the Document Object Model (DOM) [W3C 2001] to export the pmif.xml. It creates the entire document in memory, then writes it to a file. This facilitates the export because elements and attributes can be added in any order as long as they are added in the correct location. It is a relatively small file, e.g., 2-3K for the examples in Section 6, 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. Workloads begin execution on the CPU and upon completion transit to one of the other devices, then back to the CPU until completion. 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 from them 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 for the CPU, 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 each of those devices. To export those devices with a quantity greater than one, it is necessary to generate separate servers for each one, separate arcs, and calculate equal probabilities for the Transit elements from the CPU. Each of those devices returns to the CPU, so no special calculation is required for that Transit element. Several tools including Qnap have the capability to represent arrays or lists of servers, so we could have added a capability to PMIF 2.0 to accommodate them. We decided not to make this change 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. Spaces are quite common in SPE·ED names. Therefore, a simple routine was created to transform characters that are not allowed in XML IDs into an underscore character.

5.3 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 the Qnap source code and we could not implement such an interface directly. Therefore, we translated the pmif.xml file into a file in Qnap’s format to demonstrate the proof of concept.

The model translation from a pmif.xml file into a Qnap input file was done using XSLT. We generated a specific XSLT file that transforms a pmif.xml file into a file that can be directly 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 some other tools with stricter ordering in the input file, in which case two possibilities would arise: The use of DOM (as used by SPE·ED to export pmif.xml) or the use of XSLT together with a conventional programming language. The use of XSLT is fairly simple, therefore we would recommend XSLT when possible for the translation into a tool’s file format.

For the case of a real implementation (i.e., implementing an interface from the tool that would read 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 probably 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.

Some specific Qnap characteristics we had to take into account and solve when implementing the translation from the PMIF schema are:

- 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 fact that a client leaves the system is specified 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* as a hint 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, at the moment we truncate the names given in the XML file but we do not test that the names are unique. Ultimately we need a routine that does this check and generates unique names if it happens to be a repetition.
- The XML allows an identifier to start with either a letter or the character ‘_’. In Qnap the first character of an identifier must be a letter. Therefore, ultimately the identifiers have to be checked to change the character ‘_’ appearing at the beginning of an identifier for a letter (avoiding repetitions).
- Other characters that are allowed in an XML identifier (except in the first position) and not in Qnap are ‘-’, ‘.’. Clearly, these also need to be replaced (avoiding repetitions) when translating into Qnap.
- 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 of the solution.
- Solving instructions and instructions for generating results are also specified in the Qnap input file. We defined default solving instructions and default instructions for generating extra results (results that are not explicitly shown by default) in order to be able to compare results given by both tools (*SPE·ED* and Qnap).
- We have also noticed some differences in stopping conditions in the two tools which make the comparison of simulation results approximate.

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

Even though the exportation of a Qnap model into a pmif.xml is out of the scope of the prototype presented in this paper we would like to point out a few issues that we think would arise when trying to do so. One possibility from a user point of view would be to generate a pmif.xml from a Qnap input file. Another would require code changes in Qnap to generate a pmif.xml from the internal QNM information. Since we did not have access to the Qnap source code, we thought of the first possibility and the following arose:

---

Some of these points would also apply when thinking of the second possibility 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 when generating the pmif.xml.

• In Qnap, it is not explicitly said whether a workload (class in Qnap) is open or closed. It has to be detected from the collection of transit definitions for each workload to be able to specify it in the pmif schema. 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 the corresponding workload created.

• Since Qnap does not have an explicit sink node, this needs to be created when the model contains open workloads.

• It is necessary to generate separate servers for the devices that behave exactly the same since in Qnap they can be manipulated as a list.

• Some characters that are allowed in a Qnap identifier (except in the first position) and not in an ID in xml are ‘#’, ‘%’, ‘@’ and ‘!’’. Clearly, these need to be replaced (avoiding repetitions) when exporting Qnap to xml.

5.5 Importing a pmif.xml model into SPE·ED

Importing pmif.xml into SPE·ED would require another interface implemented using the DOM 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.

SPE·ED also specifies software resource requirements rather than computer (device) resource requirements. An overhead matrix specifies the computer resource requirements for each software request. For example, the software model might specify the number of database accesses, then the overhead matrix specifies the CPU and Disk requirements for each database access. PMIF 2.0 specifies service requirements for each device. So a translation would be required to convert the pmif.xml ServiceRequest specifications into appropriate internal structures in SPE·ED.

SPE·ED has a concept of facilities as collections of central server models. It would take a fair amount of analysis of the topology of models with multiple facilities to identify which devices go with which facilities. There might be a way to provide some hints, like adding an optional FacilityId to node specifications, or asking the user to identify facilities.

SPE·ED has a limit on the number of distinct devices in each facility. The quantity specification allows more actual devices. So a straightforward translation of devices would likely exceed the allowable number, and it would be necessary to determine which devices could be combined with the Quantity specification. If more devices were defined in pmif.xml than allowed by SPE·ED the model would need to be modified or it couldn’t be imported.

SPE·ED also has a special interpretation for 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.

---

5 To import software processing details one would use an SPE meta-model such as that defined in [Williams and Smith 1995] rather than a PMIF.
The (optional) NumberOfVisits would be quite useful for SPE·ED, so an import-friendly sending tool should fill them in. Some conversion of names might be required, but it isn’t likely because the XML definition is more restrictive than SPE·ED names.

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 the model (project) opened by the user. The following is an excerpt of the C++ code to create the OpenWorkload element in the pmif.xml:

```cpp
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));
// The XML Document should now own the element.
SAFERELEASE(element);
addTransit(p1, cpuName, 1.0);
SAFERELEASE(p1);
```

An earlier example showed the resulting pmif.xml for this OpenWorkload element. Appendix B shows the complete pmif.xml file that was generated for the ATM example described in [Smith and Williams 1999].

6.2 Prototype: PMIF.xml to Qnap

The Qnap prototype uses XSLT to transform the pmif.xml file into a text file in Qnap’s format. The following is an excerpt of the XSLT code used to transform the pmif.xml specification of the servers and its attributes to Qnap’s 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)
```
/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>
```
6.3 Prototype Validation

We conducted tests on the ATM model from [Smith and Williams 1999] and most of the models covered in the book [Smith and Williams 2001]. The book models excluded represented software systems with major performance problems and the system execution model for them had saturated devices and was thus unstable. Early tests confirmed that the transferred model was unstable, and we eliminated those models from further study.

Table 2 shows 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. Two sets of simulation results are reported, the first is for a run of 5,000 simulated seconds, the second for 50,000 simulated seconds. Note that the SPE·ED run time is the same as Qnap, however it is reported as less because results are automatically reset after a start interval, and it stops upon completion of a batch window rather than exactly at the completion time. 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 3 sets of results are for the ATM example from [Smith and Williams 1999]. It is an open model with 2 workload classes, Withdrawal and GetBalance. The results are much closer for the longer simulation run. Note that SPE·ED does not solve multi-class models analytically; the third set of 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.
<table>
<thead>
<tr>
<th>Model Study</th>
<th>Response Time</th>
<th>CPU Utilization</th>
<th>Disk Utilization</th>
<th>Confidence / SimTime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPE-ED Qnap</td>
<td>SPE-ED Qnap</td>
<td>SPE-ED Qnap</td>
<td>SPE-ED Qnap</td>
</tr>
<tr>
<td>1. ATM (S 5000)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Withdrawal</td>
<td>11.782 12.14</td>
<td>0.006 0.002</td>
<td>0.0063 0.0025</td>
<td>0.401 0.145 0.3966 0.1524</td>
</tr>
<tr>
<td>GetBalance</td>
<td>6.155 6.453</td>
<td>0.003 0.0025</td>
<td>0.0063 0.0025</td>
<td></td>
</tr>
<tr>
<td>2. ATM (S 50000)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Withdrawal</td>
<td>11.971 11.9</td>
<td>0.006 0.003</td>
<td>0.0063 0.0025</td>
<td>0.403 0.151 0.3984 0.1519</td>
</tr>
<tr>
<td>GetBalance</td>
<td>6.354 6.362</td>
<td>0.006 0.0025</td>
<td>0.0063 0.0025</td>
<td></td>
</tr>
<tr>
<td>3. ATM (A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Withdrawal</td>
<td>11.9 6.336</td>
<td>0.0063 0.0025</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>GetBalance</td>
<td>6.336 0.0025</td>
<td>0.0063 0.0025</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>4. Drawmod3.1 (A)</td>
<td>8.513 8.486</td>
<td>0 0.00005</td>
<td>0.05 0.047</td>
<td></td>
</tr>
<tr>
<td>5. Drawmod3.2 (A)</td>
<td>8.513 12.49</td>
<td>0 0.00049</td>
<td>0.47 0.447</td>
<td></td>
</tr>
<tr>
<td>6. Drawmod3.2 (A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distinct disks</td>
<td>12.55 12.49</td>
<td>0 0.00049</td>
<td>0.45 0.447</td>
<td></td>
</tr>
<tr>
<td>7. Drawmod3.2 (S 5000)</td>
<td>12.71 11.73</td>
<td>0 0.00047</td>
<td>0.445 0.424</td>
<td>0.370 4429 95% 50000</td>
</tr>
<tr>
<td>Distinct disks</td>
<td>12.6 12.7</td>
<td>0 0.00049</td>
<td>0.45 0.4465</td>
<td>29 49585 95% 50000</td>
</tr>
<tr>
<td>9. POTS2 (A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CallOrigination</td>
<td>0.4773</td>
<td>0.369</td>
<td>0.365 48999 95% 50000</td>
<td>LineIF 0.135 0.09 0.015 0.03</td>
</tr>
<tr>
<td>CallTermination</td>
<td>0.2883</td>
<td>0.221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HangUpCalled</td>
<td>0.063 0.0996</td>
<td>0.05 0.077</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HangUpCaller</td>
<td>0.063 0.0996</td>
<td>0.05 0.077</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. POTS2 (S 5000)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CallOrigination</td>
<td>0.3501 0.5134</td>
<td>0.315 0.374</td>
<td>0.132 0.137</td>
<td>LineIF 0.132 0.137</td>
</tr>
<tr>
<td>CallTermination</td>
<td>0.2409 0.3002</td>
<td>0.219 0.222</td>
<td>0.089 0.091</td>
<td></td>
</tr>
<tr>
<td>HangUpCalled</td>
<td>0.0607 0.0645</td>
<td>0.049 0.05</td>
<td>0.015 0.015</td>
<td></td>
</tr>
<tr>
<td>HangUpCaller</td>
<td>0.0942 0.1033</td>
<td>0.077 0.078</td>
<td>0.03 0.03</td>
<td></td>
</tr>
<tr>
<td>11. WasteBucks (A)</td>
<td>14.906 14.69</td>
<td>0.02 0.017</td>
<td>0.05 0.054</td>
<td></td>
</tr>
<tr>
<td>12. WasteBucks (S 5000)</td>
<td>14.617 15.44</td>
<td>0.02 0.017</td>
<td>0.06 0.056</td>
<td>269 3893 95% 5000</td>
</tr>
<tr>
<td>13. WasteBucks (S 50000)</td>
<td>14.71 15.26</td>
<td>0.017 0.0177</td>
<td>0.0530 0.0560</td>
<td>2959 49000 95% 50000</td>
</tr>
</tbody>
</table>
The results for Model study 4 are for 1 user with a think time of 60 secs. (because it would otherwise be unstable), whereas Model studies 5 through 8 are for 10 users with a think time of 60 sec. Model study 5 shows a significant difference in results. It was determined that this was due to an inconsistency in the way SPE\textsuperscript{ED} handled multiservers: the analytical (MVA) solution treated the disks as multiservers whereas the simulation solution treated them as multiple independent servers with equally distributed load. These results were so different, that SPE\textsuperscript{ED} was modified to solve analytical models as multiple independent servers to that it will match the simulation solution. In order to compare results before the change, we generated a model with distinct disks and solved it analytically and with simulation. Those results are in model studies 6 through 8. The difference between studies 7 and 8 show the value of the longer simulation run.

Model studies 9 through 11 are for the revised version of the telephone switching example (POTS) in chapter 14 of [Smith and Williams 2001]. It is a multiclass open model. Study 9 shows the analytical results from Qnap. Studies 10 and 11 are from the shorter and longer simulation runs. There are no disks in this model, the disk column instead reports results for the Line Interface.

Model studies 11 through 13 are for the WasteBucks example in chapter 6 of [Smith and Williams 2001] - - the revised alternative that processes orders as batches (but without the synchronization in the last version). It is an open model with a single class.

These results confirm that the pmif.xml successfully transfers models between the two tools. The comparison of solutions led to the discovery and correction of the inconsistency of SPE\textsuperscript{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 quite valuable both for tool developers and for users.

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 of the 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\textsuperscript{6}, e.g., EXP, HEXP, CST, Erlang, Uniform, Cox

Several other features are available in Qnap and other similar tools, such as maximum queue capacity, semaphores (events), flags, workload phases and phase changes, mailboxes, passive resource queues, and other advanced features, would be useful in a future version of PMIF. Further study is required to determine the best way to represent these in the meta-model before they can be added to the schema.

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. Our prototypes used default instructions for the execution of the model. Further study is required to determine what options should be provided and how to represent them in the meta-model. For instance, to check scalability, we might want to run the pmif.xml model varying the number of initial clients from 1 to max. We also used default stopping conditions for the simulation runs. Specifications for stopping conditions are desirable. A run length specification seems universal, but it may be difficult to find more advanced

\textsuperscript{6} PMIF 2.0 assumes service distributions that permit exact, analytic solutions. They are not explicit in the meta-model.
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.). If is not specified, Qnap decides what it thinks is the best choice. So a specification for the solution method should be included as an optional attribute in the appropriate element for model solution specifications.

A specification of the results desired could be part of the PMIF meta-model and schema or a separate meta-model and schema could be defined for them. A comma separated text file of those results would be convenient for importing into spreadsheets, databases or other tools. Further study is required to determine what options should be provided and how to represent them in the meta-model(s). We do not want to produce everything possible because too many results are almost as bad as too few. Results should be customized to the problem being solved.

The choices for each of these extensions depend on how one wants to use the PMIF. For example, 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 just need to get the model into another tool and the user can then fill in missing details, such as solution and results specifications. 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.

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, etc.

9 Summary and 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 describing 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 for those who would like to use it.

10 References


APPENDIX A: XML SUMMARY

Extensible Markup Language (XML) is a subset of the Standard Generalized Markup Language (SGML) which describes a class of data objects called XML documents. XML was originally designed to meet the challenges of large-scale electronic publishing. These days it is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

XML can use a Document Type Definition (DTD) or an XML Schema to describe the structure of XML documents. Validation is the main benefit of their use since it can reduce the amount of error handling that must be built into the application. DTD preceded XML Schema, although, DTD has limitations that make it not suitable for many web applications, for instance it does not use XML syntax.

On the other hand, XML Schema is itself an XML vocabulary for describing XML instance documents. A schema describes a class of documents, and the term instance can be used to describe documents belonging to a specific class. Since a schema definition is an XML document, it can be processed using standard XML tools and services such as DOM and XSLT, which are introduced in the following paragraph.

The Document Object Model (DOM) defines the logical structure of HTML and XML documents, a set of interfaces to create and access the document content, and a standard set of interfaces to access and manipulate the document structure.

XSLT (eXtensible Stylesheet Language for Transformations) is a transformation language for XML documents. XSLT is designed for use as part of XSL (eXtensible Stylesheet Language), which is a stylesheet language for XML. XSLT is also designed to be used independently of XSL.
APPENDIX B: PMIF XML SCHEMA

The current version of the schema will be maintained at www.perfeng.com/pmif/pmifschema.xsd.

<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by C Smith (L&S) -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- Global -->
  <xsd:element name="QueueingNetworkModel" type="QNMTypetype=""
  <xsd:complexType name="QNMTypetype="">
    <xsd:element name="Node" type="NodeType" maxOccurs="unbounded"/>
    <xsd:element name="Arc" type="ArcType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Workload" type="WorkloadType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="ServiceRequest" type="ServiceRequestType" maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
<xsd:complexType name="WorkloadType">
<xsd:element name="SinkNode" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:attribute name="Name" type="xsd:ID" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:choice>
</xsd:complexType>
</xsd:complexType name="ServiceRequestType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="TimeServiceRequest" type="TimeServType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="DemandServiceRequest" type="DemandServType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="WorkUnitServiceRequest" type="WorkUnitServType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:choice>
</xsd:complexType>
</xsd:complexType name="DemandServType">
  <xsd:sequence>
    <xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="WorkloadName" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="ServerID" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
  <xsd:attribute name="ServiceDemand" type="nonNegativeFloat" use="required"/>
  <xsd:attribute name="NumberOfVisits" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>
</xsd:complexType name="TimeServType">
  <xsd:sequence>
    <xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="WorkloadName" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="ServerID" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
  <xsd:attribute name="ServiceTime" type="nonNegativeFloat" use="required"/>
  <xsd:attribute name="NumberOfVisits" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>
</xsd:complexType name="WorkUnitServType">
  <xsd:sequence>
    <xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="WorkloadName" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="ServerID" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="NumberOfVisits" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>
</xsd:complexType name="TransitType">
  <xsd:attribute name="To" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="Probability" type="nonNegativeFloat" use="required"/>
</xsd:complexType>
</xsd:complexType name="ArcType">
  <xsd:attribute name="Description" type="xsd:string"/>
  <xsd:attribute name="FromNode" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="ToNode" type="xsd:IDREF" use="required"/>
</xsd:complexType>
</xsd:complexType>

<!-- Simple Type Definitions -->
<xsd:simpleType name="nonNegativeFloat">
  <xsd:restriction base="xsd:float">
    <xsd:minInclusive value="0.0"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:complexType>
<xsd:simpleType name="TimeUnitsType">
  <xsd:annotation>
    <xsd:documentation>
      If time units are omitted, all specifications are assumed to be the same relative units.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="day"/>
    <xsd:enumeration value="Day"/>
    <xsd:enumeration value="hr"/>
    <xsd:enumeration value="Hr"/>
    <xsd:enumeration value="min"/>
    <xsd:enumeration value="Min"/>
  </xsd:restriction>
</xsd:simpleType>
Entities (Elements):

Arc: An Arc connects two Nodes in a QueueingNetworkModel.Traversal of an Arc represents completion of a service request at the FromNode and a new request for service at the ToNode.

ClosedWorkload: A ClosedWorkload is a Workload with a fixed population that circulates among the Servers.

DemandServiceRequest: A DemandServiceRequest specifies the average service demand (service time multiplied by number of visits) provided for each workload that visits the Server.

Node: A Node represents an entity in the QueueingNetworkModel of the execution environment that either provides service or designates model topology.

Non-ServerNode: A Non-ServerNode represents a Node of the execution environment that designates model topology but does not provide processing service.

OpenWorkload: An OpenWorkload is 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.

QueueingNetworkModel: A QueueingNetworkModel represents a network of connected servers that provides processing service for Workloads.

Server: A Server represents a Node of the execution environment that provides some processing service.

ServiceRequest: A ServiceRequest specifies either the average TimeService or DemandService provided for each workload that visits the Server.

SinkNode: A SinkNode represents a Node of the execution environment that designates where OpenWorkloads terminate.

SourceNode: A SourceNode represents a Node of the execution environment that designates where OpenWorkloads originate.

TimeServiceRequest: A TimeServiceRequest specifies the average service time and number of visits provided for each workload that visits the Server.

Workload: A Workload represents a collection of transactions or jobs that make similar service requests from servers in the QueueingNetworkModel.

WorkUnitServer: A WorkUnitServer represents a Server that has the same ServiceTime for all Workloads.

WorkUnitServiceRequest: A WorkUnitServiceRequest specifies the number of visits to a WorkUnitServer.

Attributes:

ArrivalRate: The average rate at which transactions or jobs arrive from the outside world, receive service, and exit.

ArrivesAt: The Name of the SourceNode of an OpenWorkload.

DepartsAt: The Name of the SinkNode of an OpenWorkload.

FromNode: The Name of the origin Node of an Arc.

NumberOfJobs: The fixed population that circulates among the Nodes.

NumberOfVisits: The average number of visits to a Server in a ServiceRequest.

Quantity: The number of instances of a given Server. Multiple servers have one queue for service requests.

SchedulingPolicy: The policy used to select the next ServiceRequest to be served from a queue.

ServiceDemand: The total demand for a service request. Demand is the product of ServiceTime and NumberofVisits.

ServiceTime: The amount of time required for a server to perform one unit of service. A unit of service is the amount provided for each visit to the Server.

ThinkTime: The average interval of time that elapses between the completion of a transaction or job and the submission of the next transaction or job.

TimeUnits: The unit of time specified in a ServiceRequest or Workload intensity. If time units are omitted, all specifications are assumed to be the same relative units.

ToNode: The Name of the destination Node of an Arc.
APPENDIX C: PMIF.XML EXAMPLE

This example shows the pmif.xml that was generated by SPE-ED for the ATM example in [Smith and Williams 1999].

<!-- edited with XMLSpy v2004 rel. 3 U (http://www.xmlspy.com) by C Smith (L&S) -->
<QueueingNetworkModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="D:\PMIF\pmifSchema.xsd" Name="ATM Orig PMIF Example">
  <Node>
    <SourceNode Name="SourceNode"/>
    <SinkNode Name="SinkNode"/>
    <Server Name="CPU" Quantity="1" SchedulingPolicy="PS" ServiceTime="1"/>
    <WorkUnitServer Name="ATM" Quantity="1" SchedulingPolicy="IS" TimeUnits="sec" ServiceTime="1"/>
    <WorkUnitServer Name="DISKS" Quantity="1" SchedulingPolicy="FCFS" TimeUnits="sec" ServiceTime="0.05"/>
  </Node>
  <Arc FromNode="SourceNode" ToNode="CPU"/>
  <Arc FromNode="CPU" ToNode="SinkNode"/>
  <Arc FromNode="ATM" ToNode="CPU"/>
  <Arc FromNode="CPU" ToNode="ATM"/>
  <Arc FromNode="DISKS" ToNode="CPU"/>
  <Arc FromNode="CPU" ToNode="DISKS"/>
  <Workload>
    <OpenWorkload WorkloadName="Withdrawal" ArrivalRate="1" ArrivesAt="SourceNode" DepartsAt="SinkNode" TimeUnits="sec">
      <Transit To="CPU" Probability="1"/>
    </OpenWorkload>
    <OpenWorkload WorkloadName="Get_balance" ArrivalRate="1" ArrivesAt="SourceNode" DepartsAt="SinkNode" TimeUnits="sec">
      <Transit To="CPU" Probability="1"/>
    </OpenWorkload>
  </Workload>
  <ServiceRequest>
    <DemandServiceRequest WorkloadName="Withdrawal" ServerID="CPU" ServiceDemand="0.0063" TimeUnits="sec" NumberOfVisits="20">
      <Transit To="ATM" Probability="0.55"/>
      <Transit To="DISKS" Probability="0.4"/>
      <Transit To="SinkNode" Probability="0.05"/>
    </DemandServiceRequest>
    <WorkUnitServiceRequest WorkloadName="Withdrawal" ServerID="ATM" NumberOfVisits="11">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="Withdrawal" ServerID="DISKS" NumberOfVisits="8">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <DemandServiceRequest WorkloadName="Get_balance" ServerID="CPU" ServiceDemand="0.0025" TimeUnits="sec" NumberOfVisits="10">
      <Transit To="ATM" Probability="0.6"/>
      <Transit To="DISKS" Probability="0.3"/>
      <Transit To="SinkNode" Probability="0.1"/>
    </DemandServiceRequest>
    <WorkUnitServiceRequest WorkloadName="Get_balance" ServerID="ATM" NumberOfVisits="6">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="Get_balance" ServerID="DISKS" NumberOfVisits="3">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
  </ServiceRequest>
</QueueingNetworkModel>
APPENDIX D: XSLT FILE FOR QNAP IMPORT

<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Catalina M. Lladó (Universitat de les Illes Balears) -->
<!-- Transformation file from pmif.xml to a Qnap’s input text file. The model will be solved analytically -->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:variable name="SinkNodeName" select="QueueingNetworkModel/Node/SinkNode/@Name"/>

    /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>;
    QUEUE <xsl:for-each select="QueueingNetworkModel/Node/WorkUnitServer"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="substring(@Name,1,8)"/> </xsl:for-each>;
    <xsl:if test="count(QueueingNetworkModel/Workload/OpenWorkload) != 0">
      QUEUE <xsl:for-each select="QueueingNetworkModel/Workload/OpenWorkload"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="concat(substring(@ArrivesAt,1,7),string(position()))"/> </xsl:for-each> ;
    </xsl:if>
    CLASS <xsl:for-each select="QueueingNetworkModel/Workload/ClosedWorkload"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="substring(@WorkloadName,1,8)"/> </xsl:for-each>
    <xsl:for-each select="QueueingNetworkModel/Workload/OpenWorkload"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="substring(@WorkloadName,1,8)"/> </xsl:for-each>;
    REAL <xsl:for-each select="QueueingNetworkModel/Workload/OpenWorkload"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/> </xsl:for-each>
    <xsl:for-each select="QueueingNetworkModel/Workload/ClosedWorkload"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/> </xsl:for-each>;
    <xsl:for-each select="QueueingNetworkModel/Node/Server"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="substring(@Name,1,8)"/> /STATION/ NAME = <xsl:value-of select="substring(@Name,1,8)"/>;
      <xsl:choose>
        <xsl:when test='@SchedulingPolicy = "IS"'>
          TYPE = INFINITE;
          <xsl:when test='@Quantity > 1'>
            TYPE = MULTIPLE(<xsl:value-of select="@Quantity"/>);
          </xsl:when>
          SCHED = PS;
        </xsl:when>
        <xsl:when test='@SchedulingPolicy = "PS"'>
          TYPE = INFINITE;
          SCHED = PS;
        </xsl:when>
        <xsl:when test='@SchedulingPolicy = "FCFS"'>
          SCHED = FIFO;
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
    <xsl:for-each select="QueueingNetworkModel/Node/WorkUnitServer"> <xsl:if test="position() != 1">, </xsl:if> <xsl:value-of select="substring(@Name,1,8)"/> /STATION/ NAME = <xsl:value-of select="substring(@Name,1,8)"/>;
      <xsl:choose>
        <xsl:when test='@SchedulingPolicy = "IS"'>
          TYPE = INFINITE;
          <xsl:when test='@Quantity > 1'>
            TYPE = MULTIPLE(<xsl:value-of select="@Quantity"/>);
          </xsl:when>
        </xsl:when>
        <xsl:when test='@SchedulingPolicy = "PS"'>
          SCHED = PS;
        </xsl:when>
        <xsl:when test='@SchedulingPolicy = "FCFS"'>
          SCHED = FIFO;
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
SCHED = PS;
<xsl:when test='@SchedulingPolicy = "FCFS"'>
SCHED = FIFO;
</xsl:when>
<xsl:choose>
<xsl:for-each select="QueueingNetworkModel/Workload/ClosedWorkload">
/STATION/  NAME = <xsl:value-of select="substring(@ThinkDevice,1,8)"/>
INIT(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= <xsl:value-of select="@NumberOfJobs"/>
SERVICE(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= EXP(<xsl:value-of select="1 div (number(@ThinkTime))"/>
TRANSIT(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= <xsl:for-each select="Transit">
<xsl:if test="position() != 1">
, </xsl:if>
<xsl:choose>
<xsl:when test='@To = $SinkNodeName '> OUT </xsl:when>
<xsl:otherwise>
<xsl:value-of select="substring(@To,1,8)"/>
</xsl:otherwise>
</xsl:choose>, <xsl:value-of select="@Probability"/>
</xsl:for-each> ;
</xsl:for-each>
</xsl:choose>
</xsl:for-each>
<xsl:for-each select="QueueingNetworkModel/Workload/OpenWorkload">
/STATION/  NAME = <xsl:value-of select="concat(substring(@ArrivesAt,1,7),string(position())))"/>
TYPE= SOURCE;
SERVICE= EXP(<xsl:value-of select="1 div (@ArrivalRate)"/>
TRANSIT= <xsl:for-each select="Transit"> <xsl:value-of select="substring(@To,1,8)"/>
</xsl:for-each>, <xsl:value-of select="substring(@WorkloadName,1,8)"/>
</xsl:for-each> ;
</xsl:for-each>
<xsl:for-each select="QueueingNetworkModel/ServiceRequest/WorkUnitServiceRequest">
/STATION/  NAME = <xsl:value-of select="substring(@ServerID,1,8)"/>
TRANSIT(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= <xsl:for-each select="Transit">
<xsl:if test="position() != 1">
, </xsl:if>
<xsl:choose>
<xsl:when test='@To = "$SinkNodeName"'> OUT </xsl:when>
<xsl:otherwise>
<xsl:value-of select="substring(@To,1,8)"/>
</xsl:otherwise>
</xsl:choose>, <xsl:value-of select="@Probability"/>
</xsl:for-each> ;
</xsl:for-each>
<xsl:for-each select="QueueingNetworkModel/ServiceRequest/DemandServiceRequest">
/STATION/  NAME = <xsl:value-of select="substring(@ServerID,1,8)"/>
SERVICE(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= EXP(<xsl:value-of select="(number(@ServiceDemand)) div (number(@NumberOfVisits))"/>
TRANSIT(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= <xsl:for-each select="Transit">
<xsl:if test="position() != 1">
, </xsl:if>
<xsl:choose>
<xsl:when test='@To = "$SinkNodeName"'> OUT </xsl:when>
<xsl:otherwise>
<xsl:value-of select="substring(@To,1,8)"/>
</xsl:otherwise>
</xsl:choose>, <xsl:value-of select="@Probability"/>
</xsl:for-each> ;
</xsl:for-each>
<xsl:for-each select="QueueingNetworkModel/ServiceRequest/TimeServiceRequest">
/STATION/  NAME = <xsl:value-of select="substring(@ServerID,1,8)"/>
SERVICE(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= EXP(<xsl:value-of select="(number(@ServiceTime))"/>
TRANSIT(<xsl:value-of select="substring(@WorkloadName,1,8)"/>)= <xsl:for-each select="Transit">
<xsl:if test="position() != 1">
, </xsl:if>
<xsl:choose>
<xsl:when test='@To = "$SinkNodeName"'> OUT </xsl:when>
<xsl:otherwise>
<xsl:value-of select="substring(@To,1,8)"/>
</xsl:otherwise>
</xsl:choose>, <xsl:value-of select="@Probability"/>
</xsl:for-each> ;
</xsl:for-each>
</xsl:for-each>
<xs:otherwise>
  <xs:value-of select="substring(@To,1,8)"/>
</xs:otherwise>
</xs:choose>, <xs:value-of select="@Probability"/>
</xs:for-each> ;
</xs:for-each>

/CONTROL/ CLASS = ALL QUEUE;

/EXEC/ BEGIN
SOLVE;

<xsl:for-each select="QueueingNetworkModel/ServiceRequest/WorkUnitServiceRequest">
  PRINT ("RESIDENCE TIME FOR  <xs:value-of select="substring(@WorkloadName,1,8)"/> , <xs:value-of select="substring(@ServerID,1,8)"/> = ", MCUSTNB(<xs:value-of select="substring(@ServerID,1,8)"/>)/MTHRUPUT(<xs:value-of select="substring(@ServerID,1,8)"/>),<xs:value-of select="substring(@WorkloadName,1,8)"/>); 
  
  <xs:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/> := <xs:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/> + MCUSTNB(<xs:value-of select="substring(@ServerID,1,8)"/>),<xs:value-of select="substring(@WorkloadName,1,8)"/>); 
</xs:for-each>

<xsl:for-each select="QueueingNetworkModel/ServiceRequest/DemandServiceRequest">
  PRINT ("RESIDENCE TIME FOR  <xs:value-of select="substring(@WorkloadName,1,8)"/> , <xs:value-of select="substring(@ServerID,1,8)"/> = ", MCUSTNB(<xs:value-of select="substring(@ServerID,1,8)"/>)/MTHRUPUT(<xs:value-of select="substring(@ServerID,1,8)"/>),<xs:value-of select="substring(@WorkloadName,1,8)"/>); 
  
  <xs:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/> := <xs:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/> + MCUSTNB(<xs:value-of select="substring(@ServerID,1,8)"/>),<xs:value-of select="substring(@WorkloadName,1,8)"/>); 
</xs:for-each>

<xsl:for-each select="QueueingNetworkModel/Workload/OpenWorkload">
  PRINT ("RESPONSE TIME FOR <xs:value-of select="substring(@WorkloadName,1,8)"/> = ", <xs:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/>); 
</xs:for-each>

<xsl:for-each select="QueueingNetworkModel/Workload/ClosedWorkload">
  PRINT ("RESPONSE TIME FOR <xs:value-of select="substring(@WorkloadName,1,8)"/> = ", <xs:value-of select="concat(string('T'),substring(@WorkloadName,1,7))"/>); 
</xs:for-each>

END;
/END/
</xs:template>
</xs:stylesheet>
APPENDIX E: QNAP INPUT FILE Example

This example shows the Qnap input file that was generated from the pmif.xml using the XSLT for the ATM example in [Smith and Williams 1999].

/DECLARE/ QUEUE CPU;
    QUEUE ATM, DISKS;
    QUEUE SourceN1, SourceN2;
    CLASS Withdraw, Get_bala;
    REAL TWithdraw, TGet_bal;

/STATION/ NAME= CPU;
    SCHED = PS;

/STATION/ NAME = ATM;
    SERVICE = EXP(1);
    TYPE = INFINITE;

/STATION/ NAME = DISKS;
    SERVICE = EXP(0.05);
    SCHED = FIFO;

/STATION/ NAME = SourceN1;
    TYPE= SOURCE;
    SERVICE= EXP(1);
    TRANSIT= CPU, Withdraw;

/STATION/ NAME = SourceN2;
    TYPE= SOURCE;
    SERVICE= EXP(1);
    TRANSIT= CPU, Get_bala;

/STATION/ NAME = ATM;
    TRANSIT(Withdraw) = CPU, 1 ;

/STATION/ NAME = DISKS;
    TRANSIT(Withdraw) = CPU, 1 ;

/STATION/ NAME = ATM;
    TRANSIT(Get_bala) = CPU, 1 ;

/STATION/ NAME = DISKS;
    TRANSIT(Get_bala) = CPU, 1 ;

/STATION/ NAME = CPU;
    SERVICE(Withdraw) = EXP(0.000315);
    TRANSIT(Withdraw) = ATM, 0.55, DISKS, 0.4, OUT, 0.05 ;

/STATION/ NAME = CPU;
    SERVICE(Get_bala) = EXP(0.00025);
    TRANSIT(Get_bala) = ATM, 0.6, DISKS, 0.3, OUT, 0.1 ;

/CONTROL/ CLASS = ALL QUEUE;

/EXEC/ BEGIN
    SOLVE;

    PRINT ("RESIDENCE TIME FOR Withdraw, ATM = ", MCASTNB(ATM,Withdraw)/MTHRUPUT (ATM,Withdraw) );
    TWithdraw := TWithdraw + MCASTNB(ATM,Withdraw);

    PRINT ("RESIDENCE TIME FOR Withdraw,DISKS = ", MCASTNB(DISKS,Withdraw)/MTHRUPUT (DISKS,Withdraw) );
    TWithdraw := TWithdraw + MCASTNB(DISKS,Withdraw);

    PRINT ("RESIDENCE TIME FOR Get_bala, ATM = ", MCASTNB(ATM,Get_bala)/MTHRUPUT (ATM,Get_bala) );
    TGet_bal := TGet_bal + MCASTNB(ATM,Get_bala);

    PRINT ("RESIDENCE TIME FOR Get_bala,DISKS = ", MCASTNB(DISKS,Get_bala) / MTHRUPUT (DISKS,Get_bala) );
    TGet_bal := TGet_bal + MCASTNB(DISKS,Get_bala);

    PRINT ("RESPONSE TIME FOR Withdraw = ", TWithdraw);
    TGet_bal := TGet_bal / MTHRUPUT(SourceN1);

    PRINT ("RESPONSE TIME FOR Get_bala = ", TGet_bal);

    END;

/END/
APPENDIX F: EXPERIENCES BUILDING THE XML TOOLS

We found that the learning curve for XML is relatively high. The XML documentation is at www.w3.org. It was hard to find the (latest) language specifications in an understandable format. Examples tend to describe obscure things about documents in general, not the basics that you need to adapt them for a special purpose such as this. We found that the primers are best for getting started.

There are a number of free tools, but often they do not work. For example, we struggled with a schema definition for days. It seemed like it should work based on the specification document, but we tried numerous versions and could not get it to validate. Finally we tried a different tool and found that it validated fine. We found recommendations from others who have tried this before to be extremely helpful. Thus this section describes lessons learned the hard way for readers who would like to learn from our experiences.

We found that it is especially useful to have a structured editor and validator. We used the home version of xmlspy from altova.com. It is rather expensive for research purposes. We used a demonstration version that has a time limit on its use.

*SPE-ED* is implemented in C++ using the Microsoft Visual C++ development environment. There are some header files provided for the DOM for this compiler, and most browsers provide the library routines that you call for the DOM. Code examples and other information are available on the Microsoft.com web site.

There is a Java API (JAXP) for processing XML data using applications written in the Java Programming language ([http://java.sun.com/xml/jaxp/index.jsp](http://java.sun.com/xml/jaxp/index.jsp)).

The following tools have been recommended:

- [http://www-106.ibm.com/developerworks/views/xml/downloads.jsp](http://www-106.ibm.com/developerworks/views/xml/downloads.jsp). Contains free tools for developers; however, some of those tools are the ones we had difficulty using.
- [http://www.garshol.priv.no/download/xmltools](http://www.garshol.priv.no/download/xmltools) has other free tools. We didn’t try these after our experience with those above.
- [http://www.apache.org/#xerces](http://www.apache.org/#xerces) has C++ tools for Apache’s Xerces C++

We also learned the following useful things about xml schemas:

- It isn’t possible to define a complex structure unless you use types. Define complex types then use sequence, choice, etc. to combine elements of that type into the structure you need. For example, we had to define a QNMTType with the sequence of Nodes, Arcs, etc. It wasn’t possible to put the sequence specification inside the definition of QueueingNetworkModel element.

```xml
<!-- Global -->
<xsd:element name="QueueingNetworkModel" type="QNMType"/>
<!-- Complex Type Definitions -->
<xsd:complexType name="QNMType">
  <xsd:sequence>
    <xsd:element name="Node" type="NodeType" maxOccurs="unbounded"/>
    <xsd:element name="Arc" type="ArcType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Workload" type="WorkloadType" maxOccurs="unbounded"/>
    <xsd:element name="ServiceRequest" type="ServiceRequestType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Name" type="xsd:string" use="optional"/>
  <xsd:attribute name="Description" type="xsd:string" use="optional"/>
  <xsd:attribute name="Date-Time" type="xsd:dateTime" use="optional"/>
</xsd:complexType>
```
• *Sequence* requires zero or more of each of the elements in the order they are defined. *Choice* may be exactly one of the elements. *All* may contain zero or one of each element in any order (but not more than one). Note that the following example shows a trick to relax this constraint: The sequence in QNMType specifies that there must be 1 or more *Workloads*. Then the WorkloadType specifies that each workload may be a choice of either an *OpenWorkload* or a *ClosedWorkload* (not shown here).

```xml
<xsd:complexType name="QNMType">
  <xsd:sequence>
    <xsd:element name="Node" type="NodeType" maxOccurs="unbounded"/>
    <xsd:element name="Arc" type="ArcType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="Workload" type="WorkloadType" maxOccurs="unbounded"/>
    <xsd:element name="ServiceRequest" type="ServiceRequestType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Name" type="xsd:string" use="optional"/>
  <xsd:attribute name="Description" type="xsd:string" use="optional"/>
  <xsd:attribute name="Date-Time" type="xsd:dateTime" use="optional"/>
</xsd:complexType>

<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:choice>
</xsd:complexType>
```

• We used “empty content” to define an element that contains only attributes as in the following example:

```xml
<xsd:complexType name="TransitType">
  <xsd:attribute name="To" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="Probability" type="nonNegativeFloat" use="required"/>
</xsd:complexType>
```

• IDs can’t contain spaces and can’t start with numbers. The formal definition of an ID is in section 4.3.