Appears in Proc. Workshop on Software and Performance 2005

From UML models to software performance results: An SPE process based on XML interchange formats

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 Informàtica
Cra. de Valldemossa, Km 7.6
07071 Palma de Mallorca, Spain
cllado@uib.es

Vittorio Cortellessa, Antinisca Di Marco
Dipartimento di Informatica
Università dell'Aquila
Via Vetoio, Coppito
L'Aquila, 67010, Italy
cortelle@di.univaq.it
adimarco@di.univaq.it

Lloyd G. Williams
Software Engineering Research
2345 Dogwood Circle
Louisville, CO 80027
lloydw@perfx.net

ABSTRACT

The SPE process uses multiple performance assessment tools depending on the state of the software and the amount of performance data available. This paper describes two XML based interchange formats that facilitate using a variety of performance tools in a plug-and-play manner thus enabling the use of the tool best suited to the analysis. The Software Performance Model Interchange Format (S-PMIF) is a common representation that is used to exchange information between (UML-based) software design tools and software performance engineering tools. On the other hand, the performance model interchange format (PMIF 2.0) is a common representation for system performance model data that can be used to move models among system performance modeling tools that use a queueing network model paradigm. This paper first defines an XML based S-PMIF based on an updated SPE meta-model Then it demonstrates the feasibility of using both the S-PMIF and the PMIF 2.0 to automatically translate an architecture description in UML into both a software performance model and a system performance model to study the performance characteristics of the architecture. This required the implementation of some extensions to the XPRIT software in order to export UML models into the S-PMIF and a new function in the *SPE·ED* software to import S-PMIF models, which are also described. The SPE process and an experimental proof of concept are presented.

Categories and Subject Descriptors

B.8.2 [**Hardware**]: Performance Analysis and Design Aids; D.2 [**Software**]: Software Engineering; C.4 [**Performance of Systems**]: Modeling Techniques.

Keywords

Software Performance Engineering, tool interoperability, XML, performance model, UML, interchange format, automated model building, SPE process, methods and tools

1. INTRODUCTION

The SPE process uses multiple performance assessment tools depending on the state of the software and the amount of performance data available. This paper describes two XML based interchange formats that facilitate using a variety of performance tools in a plug-and-play manner, thus enabling the use of the tool best suited to the analysis. A Software Performance Model Interchange Format (S-PMIF) is a common representation that can be used to exchange information between (UML-based) software design tools and software performance engineering tools. Using it, a software tool can capture software architecture and design information along with some performance information and export it to a software performance engineering tool for model elaboration and solution without the need for laborious manual translation from one tool's representation to another, and the need to validate the resulting specification.

S-PMIF enables the following Software Performance Engineering (SPE) tasks:

1. Developers can prepare designs as they usually do and export the data to SPE tools where performance models can be constructed automatically.

2. The model transformation can be used to check that the resulting processing details are those intended by the UML specification.

3. Data available to developers can be captured in the development tool – other data can be added by performance specialists in the SPE tool.

4. Rapid production of models makes data available for supporting design decisions in a timely fashion. This is good for studying architecture and design tradeoffs before committing to code.

5. Developers can do some of this on their own without needing detailed knowledge of performance models.

The performance model interchange format (PMIF 2.0) is a common representation for system performance model data that can be used to move models among system performance modeling tools that use a queueing network model paradigm [20]. A user of several tools that support these formats can create a model in one tool and easily move models to other tools for further work.

This paper first defines an XML based S-PMIF based on the meta-model of software performance model information requirements in [22]. Then it demonstrates the feasibility of using both the S-PMIF and the PMIF 2.0 to automatically translate an architecture description in UML into both a software performance model and a system performance model to study the performance characteristics of the architecture. The software performance model provides best and worst case performance data for an architecture/design. If the predicted performance results do not meet performance requirements, the model identifies critical areas and makes it easy for an analyst to study alternatives for correcting problems and quantify the performance improvement of each. Once an appropriate architecture/design is selected, the PMIF can be used to transfer the model to a system execution model to study additional facets of the operating environment and look for problems due to contention, locking, etc., and to study the effect of changes in the computer or network environment.

This overall process is beneficial because no single tool is good for everything. Early in development one needs to quickly and easily create a simple model to determine whether a particular architecture will meet performance requirements. Precise data is not available at that time, so simple models are appropriate for identifying problem areas. Later in development, when some performance measurements are available, more detailed models such as Queueing Network Models (QNM), Stochastic Petri Nets (SPN), or Process Algebra (PA) models can be used to study intricacies of the performance of the system. At that time, different tools are desirable that provide features not in the simpler models. These "industrial strength" modeling tools are seldom appropriate earlier in development because the models take additional time and expertise to construct and evaluate, and it is seldom justified when performance details are sketchy at best.

A common set of XML based interchange formats lets one use a variety of different tools as long as they support the interchange. Each tool must either provide an explicit import and export command, or provide an interface to/from a file and an XSLT translation can convert between the interchange format and the file. The translation can be relatively easy.

Earlier work defined both a meta-model for software performance models and a PMIF using an EIA/CDIF (Electronic Industries Association/CASE Data Interchange Format) paradigm for transferring information between CASE tools [18, 22]. The PMIF was subsequently enhanced and implemented in XML [20]. An exchange takes place via a file and internal tool information is translated to and from the file's transfer format. 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 project uses the SPE meta-model as a starting point, and contributes the following to the interchange process:

- An updated SPE meta-model
- Definition of the XML schema based on the meta-model
- Implementation of extensions to the XPRIT software to export UML models into the S-PMIF
- Implementation of extensions to the *SPE·ED* software to import S-PMIF models
- Demonstrated feasibility with an experimental proof of concept that uses both interchange formats to combine the use of software performance engineering models and system performance models to predict performance from a UML specification.

After discussing related work, this paper describes the SPE meta-model and the XML schema based on it. Then it presents the SPE process for model exchanges and the required extensions to XPRIT and *SPE·ED*. The SPE process and the experimental proof of concept are presented. Plans for future work and conclusions complete the presentation.

2. RELATED WORK

In recent years, a significant amount of effort has been put into the inclusion of performance analysis and evaluation in the early stages of a software development process. A considerable part of it is summarized in [1].

Many of these works transfer design specifications into a particular solver that can be based on Queueing Networks, Petri Nets or Process Algebra formalisms and that can be solved either analytically or by simulation tools. Some examples on these lines are: Gu and Petriu use XSLT (eXtensible Stylesheet Language for Transformations) to transform UML models in XML format to the corresponding Layered Queueing Network (LQN) description which can be read directly by existing LQN solvers [5]. Marzolla and Balsamo propose a "UML Performance Simulator" which transforms a UML software specification given by a set of annotated diagrams (Use Case, Activity and Deployment), with a discrete-event simulation model [8]. Savino et al. annotate UML diagrams and transform them into the Qnap modeling language [13]. Lopez-Grao et al. propose a method to translate several UML diagram types to analyzable GSPN models where performance requirements are annotated according to the UML Profile for Schedulability, Performance and Time [7]. Petriu and Woodside translate specifications from Use Case Maps into LQN models [11].

Differently, our work follows the software performance engineering approach where from an annotated UML software specification, a software performance model is first derived and evaluated using a software modeling tool, like *SPE·ED* [16, 17], SP [6], or HIT [2], which outputs are normally enough in early stages of design. When more specific performance measures are needed, the model can be exported as a Queuing Network model and analyzed with a system modeling tool, like Qnap. Furthermore, our approach proposes and uses common XML based interchange formats, S-PMIF and PMIF 2.0, which allow multiple tools to be used to solve the models. Tools may be used in a "plug and play" fashion to select the tool best suited for a particular problem. It simplifies the implementation of an interchange process because tools only need to interface with the interchange format and need not develop custom interfaces to each other. The process that we envision is illustrated in Figure 1.

3. SPE META-MODEL

The SPE meta-model formally defines the information required to perform an SPE study. This model is known as the SPE meta-model because it is a model of the information that goes into constructing an SPE model. Note that this meta-model is different from the Performance Model Interchange Format (PMIF) discussed in [15, 18, 20]. The PMIF defines information exchanged between queueing network modeling tools (QNM) while the meta-model defines information to be exchanged between UML software design tools and performance tools. Additional information, such as the mapping of components to processing locations as well as the internal characteristics of software locations may be exchanged between UML and performance tools. This exchange may lay on PMIF or an extension of it where needed.

3.1 SPE Meta-Model 2.0

This meta-model defines the essential information required to create the software and system performance models as defined in [14, 19]. The SPE meta-model class diagram is shown in Figure 2a. Figure 2b shows the attributes of each object. (Note: Object attributes are typically defined as part of the class diagram. They are shown in Figure 2b here to conserve space.) The following paragraphs describe the classes and their relationships. The complete definition is in [21].

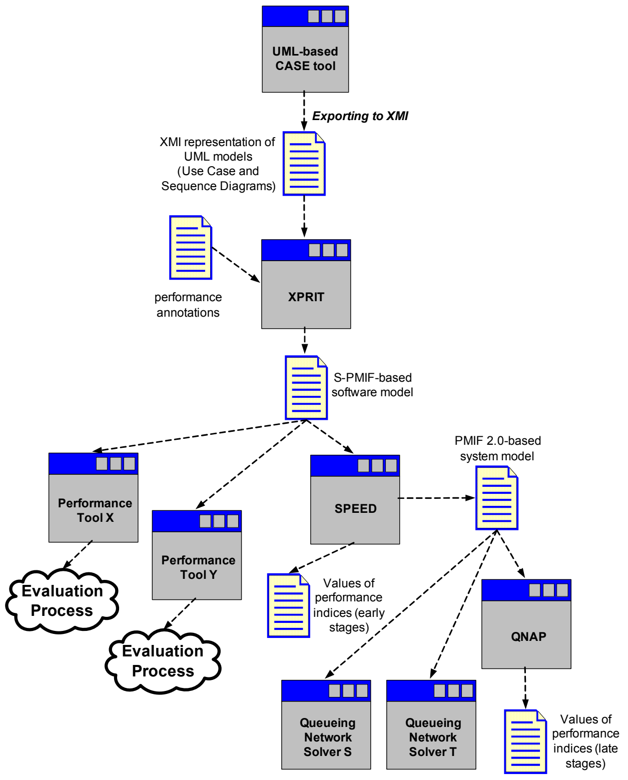
UML-based CASE tool
Exporting to XMI
XMI representation of UML models (Use Case and Sequence Diagrams)
performance annotations
XPRIT
S-PMIF-based software model
Performance Tool X
Performance Tool Y
SPEED
PMIF 2.0-based system model
Evaluation Process
Evaluation Process
Values of performance indices (early stages)
QNAP
Queueing Network Solver S
Queueing Network Solver T
Values of performance indices (late stages)

Figure 1. The SPE interchange process

An SPE study is based on *Projects* which contain one or more *PerformanceScenarios*. Each *PerformanceScenario* is modeled by an *ExecutionGraph*. An *ExecutionGraph* is composed of one or more *Nodes* and zero or more *Arcs*. A *Node* may be connected to 0, 1, or 2 other *Nodes* via an *Arc*.[*] Several types of *Nodes* may be used in constructing an *ExecutionGraph*:

ProcessingNode: represents processing steps at an appropriate level of detail. There are four types of *ProcessingNodes*:

1 *BasicNode*: represents a software processing step at the lowest level of detail appropriate for the current development stage.

2 *ExpandedNode*: indicates that processing details are expanded in a subgraph at the next level of detail. The subgraph, itself, is another *ExecutionGraph*.

3 *LinkNode*: represents a component whose execution requirements are specified in a previously saved performance scenario.

[*] Note that some *CompoundNodes* may be connected to more than 2 attached nodes, but *Arcs* are not defined for those connections. So *Nodes* can be connected to at most one predecessor and one successor *Node* by an *Arc*.

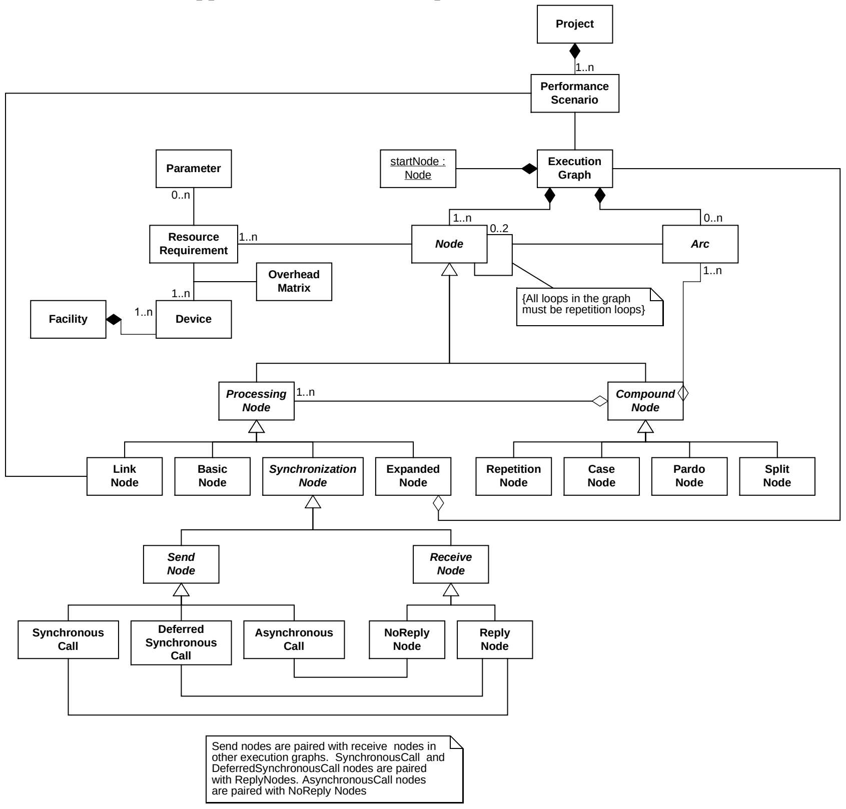

Figure 2a. SPE Meta-Model Diagram

Arc
- FromNode
- ToNode

BasicNode

CaseNode
- ArcList
- NodeList

CompoundNode

Device
- Name
- DeviceKind
- Quantity
- SchedulingPolicy
- ServiceUnits
- ServiceTime

ExecutionGraph
- Name
- Description
- ModificationDateTime
- NodeList
- ArcList
- IsMainEG

ExpandedNode
- EGName

Facility
- Name
- DeviceList

LinkNode
- PerformanceScenarioName

Node
- Name
- Type
- Probability
- Location
- ResourceRequirement

OverheadMartix
- ResourceName
- DeviceName
- AmountOfService

Parameter
- Name
- Type
- Value

PardoNode
- ArcList
- NodeList

PerformanceScenario
- Name
- InterarrivalTime
- NumberOfJobs
- Priority

ProcessingNode
- ProcessingNodeType

Project
- Name

RepetitionNode
- RepetitionFactor

ResourceRequirement
- ResourceName
- UnitsOfService

SplitNode
- ArcList
- NodeList

SynchronizationNode
- ReceiverPerfScenarioName
- Receiver
- ReceiverType

Figure 2b. Meta-model Attributes

4 *SynchronizationNode*: represents communication and synchronization with a *SynchronizationNode* in another *PerformanceScenario*. A *SynchronizationNode* may be a *SendNode* or *ReceiveNode*.

4.1 *SendNode* represents a call from one process to another. There are three types of *SendNodes*:

4.1.1 *SynchronousCall:* represents a call in which the caller waits for a reply before proceeding

4.1.2 *DeferredSynchronousCall:* represents a call in which the caller continues to execute and later requests the reply. If the reply is not available at that time then the caller waits.

4.1.3 *AsynchronousCall:* represents a call with no reply.

4.2 *ReceiveNode*: represents the receipt of a request from another process. There are 2 types of *ReceiveNodes*:

4.2.1 ReplyNode: represents receipt of request that requires a reply. It can be used with either a *SynchronousCall* or a *DeferredSynchronousCall.*

4.2.2 NoReplyNode: represents receipt of a request for which a reply cannot be sent (i.e., an *AsynchronouCall).*

CompoundNode: represents special processing structures, such as Case constructs, repetition, and parallel execution. There are four types of *CompoundNode*:

1. *RepetitionNode*: represents processing that is repeated and a repetition factor specifies the number of repetitions.

2. *CaseNode*: represents conditional execution of components, each with a probability of execution.

3. *PardoNode*: represents parallel execution paths, each with a probability of being initiated. The parallel execution paths join when they finish.

4. *SplitNode*: indicates the initiation of concurrent processes, each with a probability of being initiated, that need not join.

A *CompoundNode* is also composed of one or more *ProcessingNodes* and one or more *Arcs*.

The resources used by a *Node* are specified by one or more *ResourceRequirements*. A *ResourceRequirement* may be described by an optional *Parameter*. A *Facility* is a collection of *Devices*. A *ResourceRequirement* is executed on one or more *Devices*. A *Device* represents a unit that provides some processing service. *ResourceRequirements* are associated with *Devices* by an *OverheadMatrix* which specifies the amount of service that each resource type requires from various devices.

The current version of the meta-model does not include performance requirements. Currently, performance requirements are defined informally, based on the type of problem and expert judgment. Inclusion of performance requirements in the meta-model will require that they be more formally defined. This is a topic for future research.

The *OverheadMatrix* merits some additional explanation. It is based on a concept in [19] and the SPE product, *SPE•ED™* [16, 17] used in this demonstration. The *OverheadMatrix* is an associative entity; it describes the relationship between a *ResourceName* and a *Device*. An individual instance of *OverheadMatrix* contains a *ResourceName* a *DeviceName* and an *AmountOfService*. For example, the *ResourceRequirement* may specify the number of instructions to be executed. The *OverheadMatrix* would specify the CPU processing time per instruction as the *AmountOfService* for the CPU *Device*. The class may be viewed as a table with each instance corresponding to a row that specifies a distinct *ResourceName/DeviceName* pair such as:

- instructions and the CPU processing time per instruction,
- database updates and the CPU processing time per update
- database updates and the Disk device visits per update.

The use of the overhead matrix makes it possible to separate the portion of the model that describes the software from the portion that describes the execution environment. This is important for the SPE approach because developers are often able to specify the software resource requirements such as the number of database updates or messages transmitted, but are unable to specify the device requirements for them. The overhead matrix thus provides a mechanism to separate the two and to obtain the *ResourceName* and *UnitsOfService* from the software specification and the *OverheadMatrix* from other sources such as measurement tools or computer experts.

3.2 Adjustments to the meta-model

The following changes were made to the original SPE meta-model to reflect more recent information in [19]:

- The *StateIdentification* node was deleted and the *SynchronizationNode* was added a subclass of *ProcessingNode*
- *Facility* was added
- *Project* was added
- *Device* definitions were modified to specify the specific kind of device (such as CPU, Disk, etc.) rather than the generic terms FCFS, NonFCFSDemandSpec, and NonFCFSTimeSpec.

Other minor changes were made to class attributes for the XML implementation. For example, XML schemas allow names to be used as IDs and ID references, so NodeIds were eliminated. We changed the specification for names to match XML names in http://www.w3.orgTR/2004/REC-xml-20040204/#id. Other changes are similar to those made in [20].

3.3 S-PMIF XML Schema

The diagram of a portion of the XML schema corresponding to the S-PMIF meta-model is shown in Figure 3. The complete schema is at www.perfeng.com/pmif/s-pmifschema.xsd. The following excerpt shows the schema definition for an *ExecutionGraph*:

```
<xs:complexType name="EG_type">
 <xs:sequence>
  <xs:choice maxOccurs="unbounded">
   <xs:sequence>
    <xs:choice>
     <xs:element name="BasicNode" type="BasicNode_type"/>
     <xs:element name="ExpandedNode"
    type="ExpandedNode_type"/>
     <xs:element name="LinkNode" type="LinkNode_type"/>
     <xs:element name="SynchronizationNode"
    type="SynchroNode_type"/>
    </xs:choice>
    <xs:element name="ResourceRequirement"
    type="ResourceRequirement_type" "minOccurs="0"
    maxOccurs="unbounded">
    </xs:element>
   </xs:sequence>
   <xs:element name="CompoundNode"
    type="CompoundNode_type"/>
  </xs:choice>
  <xs:element name="Arc" type="Arc_type" minOccurs="0"
    maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="EGname" type="xs:ID" use="required"/>
```

```
  <xs:attribute name="IsMainEG" type="xs:boolean"
     use="required"/>
  <xs:attribute name="StartNode" type="xs:IDREF"
     use="required"/>
  <xs:attribute name="ModificationDateTime" type="xs:dateTime"
     use="optional"/>
  <xs:attribute name="SWmodelname" type="xs:string"
     use="optional"/>
</xs:complexType>
```

A sample s-pmif.xml *ExecutionGraph* specification for this schema is in Section 5.

The schema has 2 differences from the meta-model. First, we flattened the hierarchy in several areas to simplify the xml. For example, both *Nodes* and *ProcessingNodes* are eliminated from the schema and their attributes are moved to the nodes that inherit those attributes.

Second, we made some elements and attributes optional in the schema even though they are not optional in a software performance model. For example, a workload intensity such as interarrival time is necessary to solve a software performance model; however, the developer of the UML software diagrams may not know that information so it won't be required in the xml. Similarly, we made resource requirements, overhead matrix and device specifications optional. We discuss this issue further in the next section.

We also created three separate schemas for the meta-model: Topology, Overhead_Matrix, and Device. They can be combined by including the appropriate schemas. Thus, Topology may include Overhead_Matrix which includes Device. This is useful because one may use any of the schemas without using the others. For example, if the overhead matrix specification is coming from another source it does not need to be included in the topology, and vice-versa.

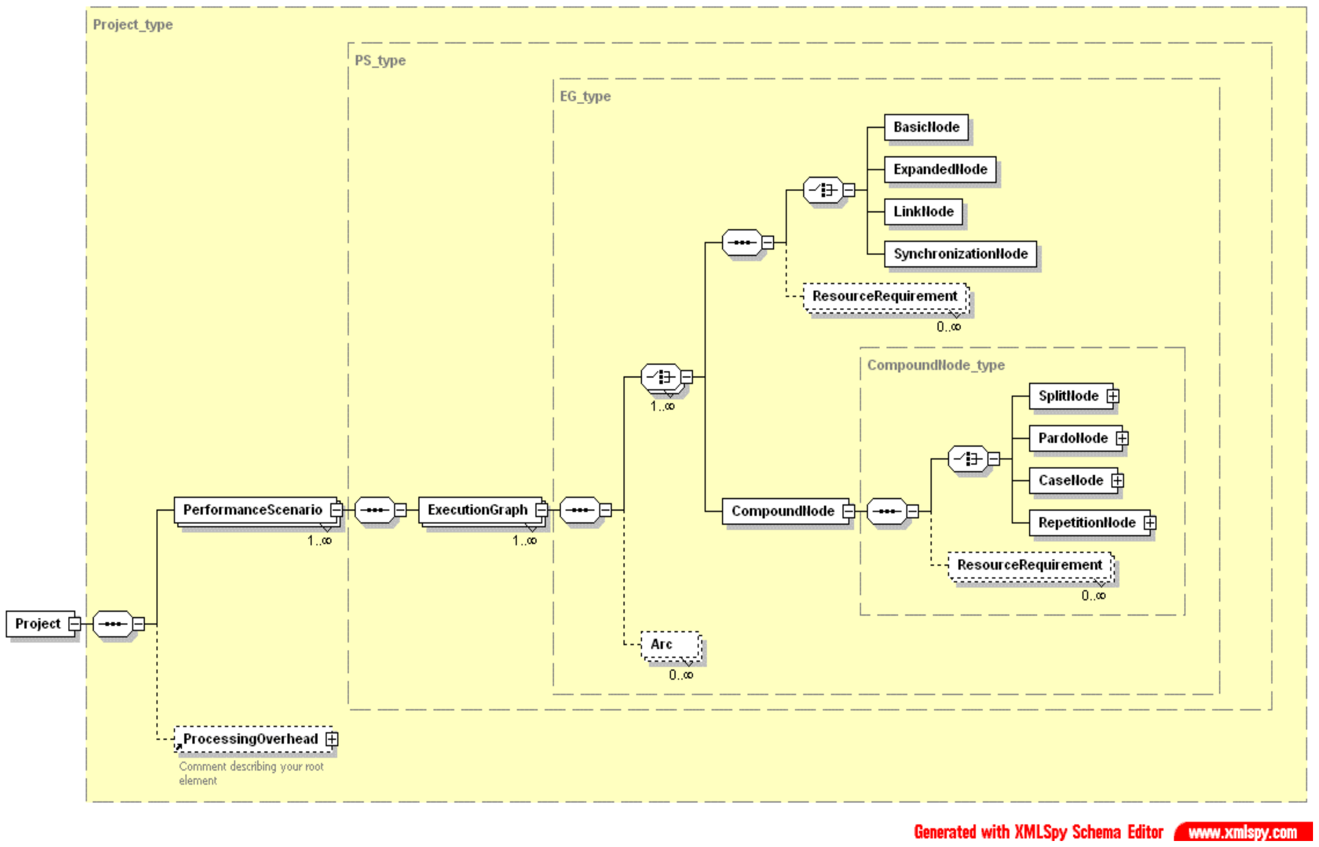

Figure 3. Portion of the XML schema corresponding to the S-PMIF meta-model

4. SPE Model Interchange Process

Our vision for the SPE model interchange process (shown in Figure 1) is:

1. A software architect, designer, or developer would use a UML tool to create their model of the software and when ready for the assessment, export the model into S-PMIF.

2. A software performance engineer would then import the S-PMIF into a software performance modeling tool such as *SPE·ED*. They would likely need to supplement the information received from S-PMIF to add one or more of the following: resource requirements, facility and device characteristics, and the overhead matrix. The latter task may be skipped when the original UML model is annotated with all the additional performance information needed (using, for example, the UML

SPT profile [10]), and the translation tool is able to process this additional information.

3. The software performance engineer would conduct performance studies, and if problems are found, modify the software performance model accordingly.

4. After resolving any serious problems with the software architecture and/or design, they may export the model into PMIF.

5. A performance engineer would import the PMIF into a system or network modeling tool for further investigation of performance properties of the network and computer system such as the effect of locking and contention with other work in the environment.

Results would then be exchanged in the reverse direction and ultimately the software specialist would be able to view suggestions for performance improvements and automatically update the UML to reflect selected changes.Note that the reverse direction is not shown in Figure 1, nor part of this work. The meta-model and schema may need some modifications if performance results or changes to the software model, such as node coordinates or other revisions, are to be retained in the UML for future evaluations.

This process differs from that proposed by other authors primarily because we envision the use of a software performance modeling tool such as *SPE·ED*, SP or HIT between the UML and the system performance modeling step using QNM, SPN, or PA. In our experience, we find many software problems that must be corrected before detailed study of the system performance is feasible. The case study described later illustrates. When problems are detected, it isn't enough to know that the system is saturated. It is also necessary to determine which parts of the software contribute to the problem and how much, in order to determine options for solving an architecture or design problem. For example, the case study has a problem due to excessive disk usage. A software performance model can identify which portions of the software use the disk and enable the evaluation of different software alternatives that use less I/O. A system performance model, however, will be limited to hardware improvement alternatives such as more or faster devices because the detailed studies require data that is typically not available until later in development. At that stage, the time and cost to change the architecture or design is prohibitive so hardware alternatives are the only options that are viable. The best solution may be a combination of software and hardware improvements. Our model interchange process enables the evaluation of all those options.

4.1 Philosophy

The model interchange strategy that we adopted from CDIF [4] is "export everything you know and provide defaults for other required information"; and "import the parts you need and make appropriate assumptions for required data that is not in the schema and thus the interchange file."

We started with a use case for the SPE interchange process in which developers did not have resource requirement specifications, the facility or device information, etc. So it was necessary to fill in many default values such as equal probabilities for Case nodes, etc.

Our PMIF experience led us to the realization that 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.

4.2 Exporting UML models to S-PMIF

This is a two-steps task: (i) exporting UML diagrams from a CASE tool representation to an XML format, (ii) transforming the exported result into a S-PMIF model.

For what concerns the first step, the XMI standard specifications [9] have been adopted by almost all UML CASE tools to export UML diagrams in XML. Actually XMI does not represent a specific Schema for UML diagrams, but gives formal specifications to build standard Schemas for UML diagrams. This is the reason for small differences among the XMI exporting results of UML tools. For the sake of this paper experiments we have used the Poseidon tool [12].

The XPRIT tool performs the second step [3]. XPRIT is made of two components: *UML2EG*, that allows to annotate Use Case and Sequence Diagrams and generate from the annotated diagrams an Execution Graph; *UML2QN*, that allows to annotate a Deployment Diagram and generate from the annotated diagram a Queueing Network representing the hardware platform where the software shall run.

For the sake of these experiments we have used only *UML2EG*, as the generation of a Queueing Network has been delayed in the process. In particular, we have exploited the XPRIT capability of producing the structure of an Execution Graph (owing S-PMIF) from one or more UML Sequence Diagrams (represented in XMI). The translation algorithm is based on visiting the Sequence Diagram and recognizing elementary patterns. For each pattern in a Sequence Diagram a corresponding pattern of an Execution Graph is associated. The whole structure of the Sequence Diagram is used to interconnect elementary patterns in the Execution Graph. For example, UML2EG is able to recognize sequential and parallel patterns, synchronous and asynchronous communications.

Some accommodations were needed on the UML diagrams to make XPRIT work on these experiments:

1. In order to avoid XPRIT considering the paths that depart one after the other from the same SD axis (see *draw()*'s leaving *Beam* in Figure 4) to all be parallel paths, we added return arrows to the diagrams;

2. XPRIT does not cope with object creation, as all the names of components acting in a diagram need to be known in advance; therefore object creation has been modeled as a standard synchronous message between two existing components;

3. Software loops are not part of the UML 1.x standards (which is the basis of XPRIT), so message labels have been exploited to delimitate the starting and the ending messages of a loop in a Sequence Diagram.

Note that the last limitation will disappear with UML 2 Sequence Diagrams, where frames have been introduced to delimitate special interaction patterns. A new XPRIT release is being implemented based on UML 2, so many translation steps will become straightforward.

4.3 Importing S-PMIF models into *SPE·ED*

SPE·ED uses the Document Object Model (DOM) to import the s-pmif.xml. It first loads and parses the document, then uses DOM calls to walk through each execution graph and create the corresponding nodes and arcs in *SPE·ED*.

SPE·ED required a custom interface because, rather than reading input from a file, it provides a graphical user interface that enables a user to quickly draw a model. When the input comes from an S-PMIF, there is currently no provision for location coordinates for the nodes. Therefore another special routine is required to "reformat" a graph and assign nodes to locations.

4.4 Exporting a pmif.xml model from *SPE·ED*

SPE·ED also uses the Document Object Model (DOM) 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 example in section 5, 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 can contain multiple facilities, each with this central service topology.

Several other cases required special handling, such as generating source, sink, and think nodes, transit probabilities, generating separate servers when quantity of servers is greater than one, name substitutions, etc. Details are in [20].

4.5 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.

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.

5. EXPERIMENTAL RESULTS

For the proof of concept we used the Drawmod Architecture 1 model described in Chapter 4 of [19]. The sequence diagram for the model is in Figure 4.

The following is part of the XML file resulting from the XPRIT translation of the sequence diagram:

```
<PerformanceScenario ScenarioName="drawmod_1"
    SWmodelfilename="drawmod_1_SD.xmi">
  <ExecutionGraph EGname="drawmod_1" IsMainEG="true"
      StartNode="create_Model">
    <BasicNode NodeName="create_Model"/>
    <BasicNode NodeName="draw_Model"/>
    <BasicNode NodeName="open"/>
    <BasicNode NodeName="find_modelID"/>
    <BasicNode NodeName="find_modelID_beams"/>
    <BasicNode NodeName="sort_beams"/>
    <CompoundNode>
      <RepetitionNode NodeName="r1">
        <ExpandedNode NodeName="e1" EGname="e1_ref"/>
      </RepetitionNode>
    </CompoundNode>
    <BasicNode NodeName="close"/>
    <Arc FromNode="create_Model" ToNode="draw_Model"/>
    <Arc FromNode="draw_Model" ToNode="open"/>
    <Arc FromNode="open" ToNode="find_modelID"/>
    <Arc FromNode="find_modelID"
      ToNode="find_modelID_beams"/>
    <Arc FromNode="find_modelID_beams"
      ToNode="sort_beams"/>
    <Arc FromNode="sort_beams" ToNode="r1"/>
    <Arc FromNode="r1" ToNode="close"/>
  </ExecutionGraph>
  <ExecutionGraph EGname="e1_ref" IsMainEG="false"
      StartNode="retrieve_beam">
      <!--Details omitted -->
  </ExecutionGraph>
</PerformanceScenario>
```

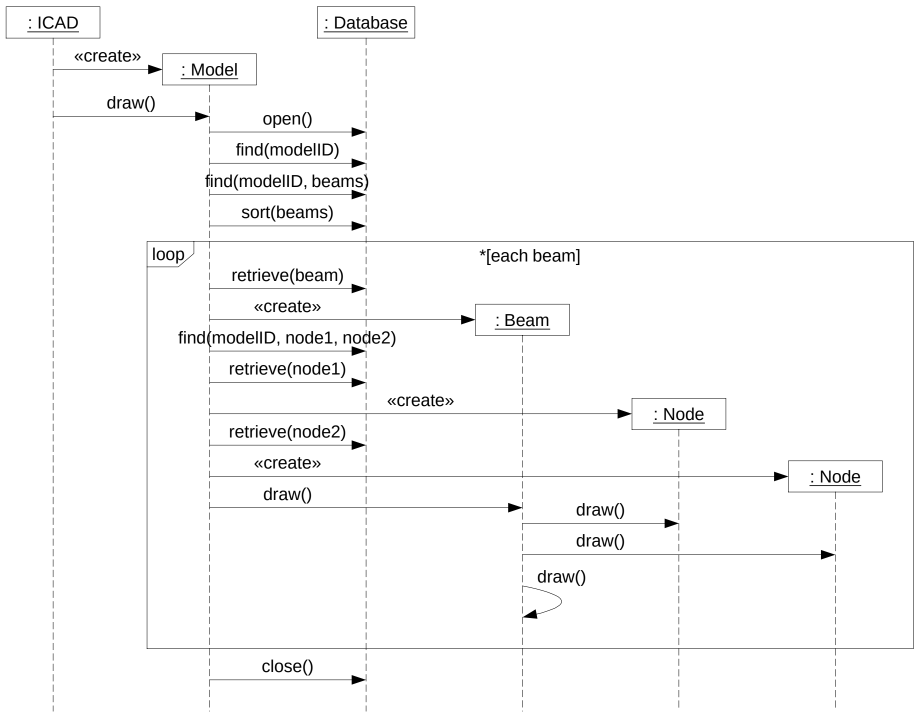

Figure 4. Drawmod Sequence diagram

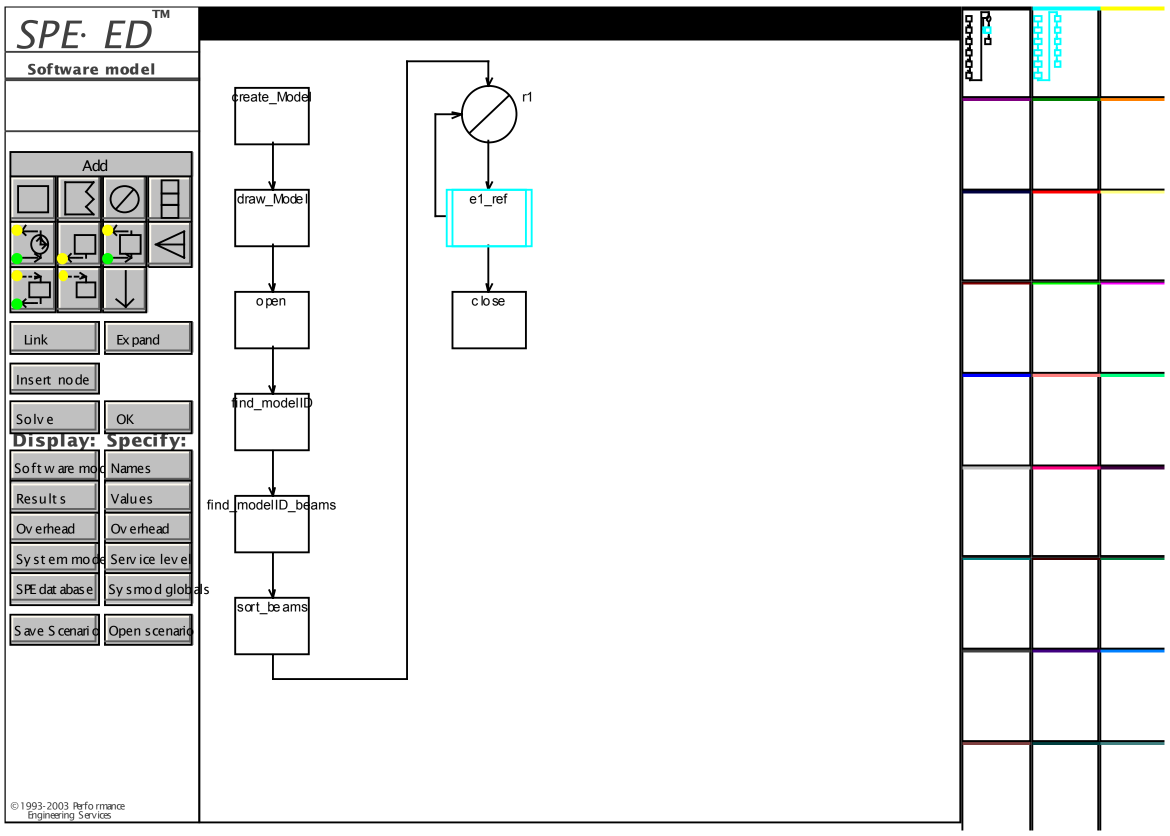

Figure 5. Generated *SPE·ED* Model

The Execution Graph has a Boolean attribute (*IsMainEG*) that indicates whether it is the main graph in the file or a sub-graph. It is followed by a sequence of nodes followed by a sequence of arcs between nodes. As long as the Sequence Diagram follows a sequential execution, all Basic Nodes are generated. Upon finding a loop, a Repetition Node is appended that refers to a subgraph identified from the *EGname* attribute "e1_ref." The complete file is in [21].

Next the s-pmif.xml model was imported into *SPE·ED* and the software model was created. The generated software model is shown in Figure 5. Note that the text does not fit into the execution graph nodes because the operating system routines use spaces to insert line breaks; however, the XML names cannot contain spaces. Some translation of names will be necessary to create "prettier" models.

Next, we added the resource requirements (from the Drawmod example in [19]), then the model was solved. In general, software performance engineers will need to use the techniques in [19] to estimate requirements that are not in the interchange file. That is an important step in the overall process, but it is beyond the scope of this paper.

The model was solved and problems were identified in the architecture. After making the architectural changes we produced Drawmod Architecture 3 (also described in [19]) and confirmed that it resolved the performance problems. Note that in this case, *SPE·ED* has the ability to solve the system execution model both analytically and with simulation to quantify the response time, utilization, etc. for computer resources so it isn't necessary to export the model to get those results. There are other reasons why one might want to export the model, such as:

- to compare solutions
- to get additional metrics such as queue lengths
- to study additional facets of the environment that might not fit the central server assumptions mentioned in section 4.4.

So the next step in the proof of concept is to export the model from *SPE·ED* into pmif.xml. The following shows an excerpt containing the generated service request (produced from *SPE·EDs* conversion of the software performance model into the system performance model):

```
<ServiceRequest>
  <DemandServiceRequest
     WorkloadName="Drawmod_Architecture_3"
     ServerID="CPU" ServiceDemand="3.574195E-03"
     TimeUnits="sec" NumberOfVisits="2219">
   <Transit To="Disk_A" Probability="4.867057E-02"/>
   <Transit To="Disk_B" Probability="4.867057E-02"/>
   <Transit To="Display" Probability="0.9022082"/>
   <Transit To="UserThink" Probability="4.506535E-04"/>
  </DemandServiceRequest>
  <WorkUnitServiceRequest
     WorkloadName="Drawmod_Architecture_3"
     ServerID="Disk_A" NumberOfVisits="108">
   <Transit To="CPU" Probability="1"/>
  </WorkUnitServiceRequest>
  <WorkUnitServiceRequest
     WorkloadName="Drawmod_Architecture_3"
     ServerID="Disk_B" NumberOfVisits="108">
   <Transit To="CPU" Probability="1"/>
  </WorkUnitServiceRequest>
  <WorkUnitServiceRequest
     WorkloadName="Drawmod_Architecture_3"
     ServerID="Display" NumberOfVisits="2002">
   <Transit To="CPU" Probability="1"/>
  </WorkUnitServiceRequest>
</ServiceRequest>
```

The pmif.xml is then imported into Qnap. In this specific implementation the import consists of an XSLT translation from a file in pmif's format into a file in Qnap's format. The generated Qnap input file for the Drawmod Architecture 3 is shown below. It can be seen that the stations need first to be declared and then they can be modified as many times as is wanted, so when reading the file sequentially, the last information read is the one that is taken. This makes the use of XSLT very convenient.

```
/DECLARE/ QUEUE UserThin, CPU;
        QUEUE Disk_A, Disk_B, Display;
        CLASS Drawmod_;
        REAL TDrawmod;

/STATION/  NAME= UserThin;
         TYPE = INFINITE;

/STATION/  NAME= CPU;
         SCHED = PS;

/STATION/  NAME = Disk_A;
         SERVICE = EXP(0.03);
         SCHED = FIFO;

/STATION/  NAME = Disk_B;
         SERVICE = EXP(0.03);
         SCHED = FIFO;

/STATION/  NAME = Display;
         SERVICE = EXP(0.001);
         TYPE = INFINITE;

/STATION/  NAME = UserThin;
         INIT(Drawmod_) = 10;
         SERVICE(Drawmod_) = EXP(60);
         TRANSIT(Drawmod_)= CPU, 1 ;

/STATION/  NAME = Disk_A;
        TRANSIT(Drawmod_) = CPU, 1 ;

/STATION/  NAME = Disk_B;
        TRANSIT(Drawmod_) = CPU, 1 ;

/STATION/  NAME = Display;
        TRANSIT(Drawmod_) = CPU, 1 ;

/STATION/  NAME = CPU;
         SERVICE(Drawmod_) =
       EXP(0.000001610723298783236);
         TRANSIT(Drawmod_) = Disk_A, 4.867057E-02,
         Disk_B, 4.867057E-02,
         Display, 0.9022082,
         UserThin, 4.506535E-04 ;
```

The Qnap model is then solved and used for further study. The results of the initial solution are reported in [20] and are not shown here. This proof of concept illustrates the feasibility of the SPE process using XML based interchange formats for using multiple tools, rather than the particular results obtained from the models.

5.1 Lessons Learned

We learned several lessons while conducting the experimental proof of concept that are described in the following paragraphs.

We found that there may be different interpretations of a UML sequence diagram and it may not be clear which is the proper interpretation. For example, the sequence of draw()s in Figure 4 were interpreted by XPRIT to be parallel steps because they did not have return arrows. We often find that, for convenience, developers do not specify return arrows from calls, and we do not want to require this specification just so the models can be exported. For this exercise, we just inserted the return arrows. In UML 2 there is a specific construct for parallel execution so this issue will no longer be a problem. In general, the interchange shows the value of viewing processing steps in different notations to confirm that the processing is specified the way the developer intended.

Note that the translated model in Figure 5 is far more detailed than the Drawmod model in Figure 4-18 of [19]. Many of the processing steps in the automatically generated model are not interesting from a performance standpoint, and the extra steps tend to "clutter" the model. This is a departure from the simple model strategy described earlier. This is a common problem with automatic translation of designs. In many cases it may be easier to just create a new model and omit those details initially. Some techniques for "pruning" an automatically generated model would make it better suited for SPE.

This proof of concept illustrates one pass from UML to Qnap. The SPE process will actually be iterative and there will be a need to exchange multiple models in the forward as well as the reverse direction. Thus, we will need to be able to retain information that was added by tools during the evaluation so that it won't have to be re-created each time, such as resource requirements, location coordinates, etc. We envision using the S-PMIF to transfer this information to the design tool where it will need to be imported, saved, and exported the next time this SPE interchange process is used.

5.2 Future Work

This work was an initial step in the overall SPE interchange process. Several additional steps are planned:

- Update XPRIT to export the new constructs in UML 2.0.
- Export resource requirements specified using the UML Profile for Schedulability, Performance and Time.
- Define a meta-model and schema for the feedback path, in order to support the transformation of "abstract" performance results into "actual" design alternatives for UML or other CASE tools.
- Define a meta-model and schema for the exchange of performance results from system performance modeling tools back to software performance engineering tools.
- Additional studies of additional models using the interchange.

6. CONCLUSIONS

This paper has described two XML interchange formats that support an SPE process that facilitates the use of the performance assessment tool best suited to the analysis task, state of the software, and amount of performance data available. It used the original SPE meta-model [22] and PMIF 2.0 [20] as a starting point and presented an updated SPE meta-model, defined an XML schema based on the meta-model, implemented extensions to the XPRIT software to export UML models into the S-PMIF, implemented extensions to the *SPE·ED* software to import S-PMIF models, and demonstrated the feasibility with an experimental proof of concept of the SPE process using multiple interchange formats and tools.

The interchange formats allow flexibility in when and how performance specifications are provided and even allow some specifications to be provided by measurement tools. The interchange also enables a "plug and play" paradigm for using performance modeling tools appropriate for the particular problem to be studied.

Using a common format simplifies the tool implementation by requiring only an import and export interface to the interchange format rather than a custom interface to each tool that exchanges information. The implementation may be done using an XSLT translation external to tools that provide a file input/output interface. Thus, users of the tool can create (and share) their own interchange mechanism when tool vendors do not provide a custom interface.

We have learned from this experience that real tool interoperability in software performance assessment can be achieved using XML technologies. The structures, the methodologies and the automatisms that we have separately defined and implemented before this experience have found here a common ground to share their potential. The use of different tools has highlighted some inconsistencies in the tools thus lead to improvements in the individual tools.

We consider this experience as a starting point to investigate the integration of CASE tools and performance tools. Internal mechanisms need certainly to be refined, and a much wider application context can be considered for tool integration. In order to widen the scope of this work, other CASE tools and performance tools can be considered, and their ability to interact through these schemas (and their evolutions) shall be studied.

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.

7. REFERENCES

[1] S. Balsamo, et al., *Model-based Performance Prediction in Software Development: A Survey*. IEEE Trans. on Software Engineering, 2004. **30**(5): p. 295-331.

[2] H. Beilner, J. Mäter, and N. Weissenburg. *Towards a Performance Modeling Environment: News on HIT*. in

Proceedings 4th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation. 1988: Plenum Publishing.

[3] V. Cortellessa, M. Gentile, and M. Pizzuti. *XPRIT: An XML-based Tool to Translate UML Diagrams into Execution Graphs and Queueing Networks (Tool Paper)*. in *Proc. of 1st Int. Conf. on the Quantitative Evaluation of Systems*. 2004. Enschede, NL: IEEE Computer Society.

[4] EIA, *CDIF - CASE Data Interchange Format Overview*, Engineering Department, Electronics Industries Association, Arlington, VA, EIA/IS-106, January 1994.

[5] G. Gu and D. Petriu. *XSLT Transformation from UML Models to LQN Performance Models*. in *Proc. Workshop on Software and Performance*. 2002. Rome: ACM.

[6] P. Hughes, *SP Principles*, STC Technology, o59/ICL226/0, July 1988.

[7] J.P. López-Grao, J. Merseguer, and J. Campos. *From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering*. in *Proc. Workshop on Software and Performance*. 2004. Redwood Shores, CA: ACM.

[8] M. Marzolla and S. Balsamo. *UML-PSI: the UML Performance Simulator (Tool paper)*. in *Proc. 1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*. 2004. Enschede, NL: IEEE Computer Society.

[9] OMG, *XML Metadata Interchange (XMI) 2.0*, OMG Full Specification, formal/03-05-02, 2002.

[10] OMG, *UML Profile for Schedulability, Performance and Time, formal/03-09-01*, OMG Full Specification, 2003.

[11] D. Petriu and C.M. Woodside. *Analyzing Software Performance Requirements Specification for Performance*. in *Proc. Workshop on Software and Performance 2002*. 2002. Rome: ACM.

[12] Poseidon, *www.gentleware.com*,

[13] N. Savino, et al. *Extending UML to Manage Performance Models for Software Architectures: A Queuing Network Approach*. in *Proc. 9th Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, SPECTS*. 2002. San Diego, CA.

[14] C.U. Smith, *Performance Engineering of Software Systems*. 1990, Reading, MA: Addison-Wesley.

[15] C.U. Smith, *Definition of A Performance Model Interchange Format*, Performance Engineering Services, PES-1001-94, October 1994.

[16] C.U. Smith and L.G. Williams, *Performance Engineering of Object-Oriented Systems with SPEED*, in *Lecture Notes in Computer Science 1245: Computer Performance Evaluation*, M.R.e. al., Editor. 1997, Springer: Berlin, Germany. p. 135-154.

[17] C.U. Smith and L.G. Williams, *Performance Engineering Evaluation of CORBA-based Distributed Systems with SPEED*, in *Lecture Notes in Computer Science*, R. Puigjaner, Editor. 1998, Springer: Berlin, Germany.

[18] C.U. Smith and L.G. Williams, *A Performance Model Interchange Format*. Journal of Systems and Software, 1999. **49**(1).

[19] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. 2002: Addison-Wesley.

[20] C.U. Smith and C.M. Lladó. *Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation*. in *Proc. 1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*. 2004. Enschede, NL: IEEE Computer Society.

[21] C.U. Smith, et al., *Software Performance Engineering Model Interchange Format (S-PMIF 2.0): XML Definition and Implementation Technical Report*, L&S Computer Technology, Inc., www.perfeng.com/paperndx.htm, Dec. 2004.

[22] L.G. Williams and C.U. Smith. *Information Requirements for Software Performance Engineering*. in *Proceedings 1995 International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*. 1995. Heidelberg, Germany: Springer.