Copyright 2005 by the authors. All rights reserved.

A PERFORMANCE MODEL WEB SERVICE

Catalina M. Lladó, Ramon Puigjaner
Universitat Illes Balears
Departament de Matemàtiques I Informàtica
Cra. de Valldemossa, Km 7.6
07071 Palma de Mallorca, Spain
cllado@uib.es, putxi@uib.es

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

Performance Engineering uses multiple performance assessment tools depending on the state of the software and the amount of performance data available. This paper demonstrates how Web Services can be used to facilitate the use of modeling tools in a plug-and-play manner thus enabling the use of the tool best suited to the analysis. It describes the design and implementation of a prototype Web Service for a performance modeling tool. Additionally, it shows experimental results that prove the viability of such a Web Service.

1. INTRODUCTION

The Software Performance Engineering (SPE) process uses multiple performance assessment tools depending on the state of the software and the amount of performance data available. Web Services are distributed software components that allow the communication among applications or application components in a standard way through common protocols that are independent of the programming language, platform, presentation format, and operating system. A Web Service is a container that encapsulates specific functions and makes these functions available to other servers.

If a software performance modeling tool, like *SPE·ED* [L&S], wants to access another performance model Web Service, the tool first exports the model to a predefined model interchange format and then makes an HTTP call (application-to-application) to this Web Service.

If the owner of another modeling tool wants to provide such a performance model Web Service the tool must provide interfaces for: importing the interchange format into the tool's model format, solving the model, and returning the results in a format that can be understood by the caller. These interfaces then need to be implemented as a Web Service as described later in the paper.

A user of several tools that support a common interchange 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 and the need to validate the resulting specification. For example, an analyst might create a model of a server platform in order to conduct several studies, and then move the model to a tool better suited to network analysis. Web Services provide a mechanism to easily invoke other tools automatically rather than having to manually execute each one. Other benefits of such a "plug and play" infrastructure include:

- Enabling users to compare solutions from multiple tools.
- A user may want to migrate a model to temporarily use another tool to develop and study more detailed models.
- A user may want to migrate a model to "permanently" use a different tool for the model.

Copyright 2005 by the authors. All rights reserved.

- 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.

- A user may want to compare different tools, for instance before buying one.

- A user may want to create input specifications in a common interchange format or in a familiar tool rather than learn the interface to multiple tools.

We could also use performance model Web Services for Software Performance Engineering (SPE). Doing so would offer the following advantages:

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; measurement data can be incorporated into the model definition; and, with the SPE tool, software performance specialists can add missing data.

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.

This work is part of an ongoing project to automate SPE performance evaluation. Previous results have established a foundation that makes implementation of a Performance model Web Service viable. These results are reviewed here and pointers are given to further information on each of them.

This paper first presents the SPE process for model exchanges and discusses the tool "plug and play" approach. It provides some background information on Web Services and how this technology can be used to implement the tool use. Then we introduce two XML based interchange formats that facilitate using a variety of performance tools, thus enabling the use of the tool best suited to the analysis. Next we describe the prototype performance model Web Service design and implementation followed by the experimental proof of concept and results. Plans for future work and conclusions complete the presentation.

2. CONCEPT

Our vision for the SPE model interchange process shown in Figure 1 is [Smith et al. 2005]:

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 a common interchange format such as Software Performance Model Interchange Format S-PMIF (described in section 4).

2. If the UML tool does not support S-PMIF an intermediate step is required to translate the UML tool format such as XMI into the model interchange format S-PMIF. The intermediate translator may also automatically import performance data from measurement tools.

3. 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 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 [OMG 2003]), and the translation tool is able to process this additional information.

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

5. After resolving any serious problems with the software architecture and/or design, they may export the model into a second interchange format, PMIF, a common representation for system performance models described as queueing networks (QNM) (described in section 4).

6. A Web Service would import the PMIF into a QNM solving 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.

Note that this description covers a first pass performance evaluation and it assumes that a

Copyright 2005 by the authors. All rights reserved.

software performance engineer initiates the use of the other tools. It is possible that the software developer using a UML tool may initiate the invocation of all the performance modeling tools, or even a system performance specialist or capacity planner may initiate the use of other tools to collect information on new software systems under development. The use of Web Services for the various tools makes any of these scenarios viable.

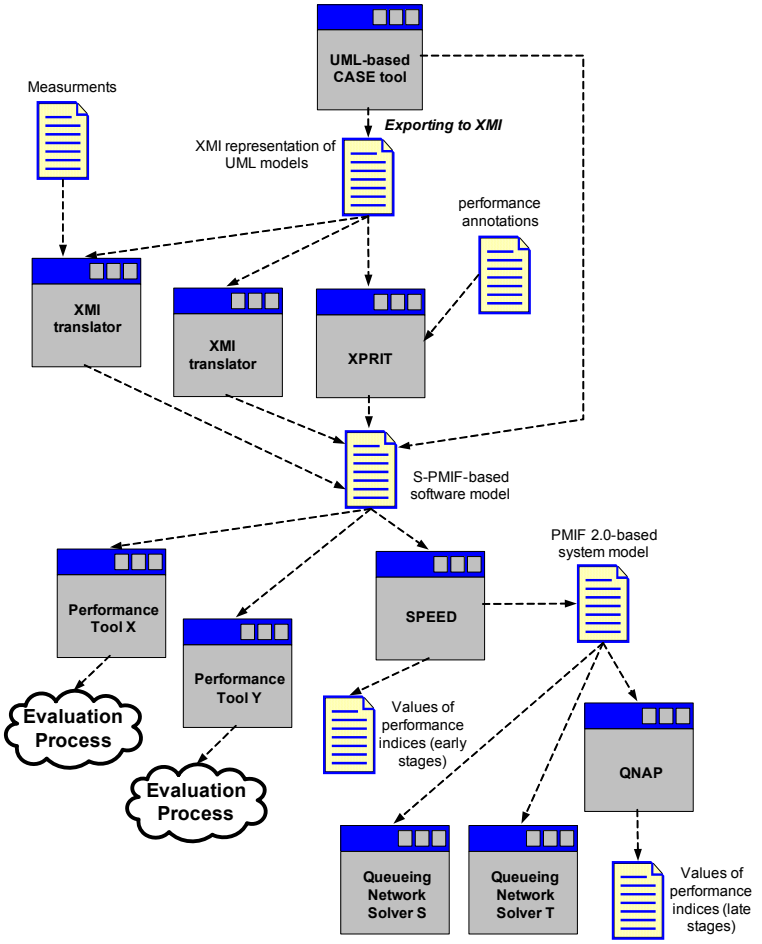

Figure 1. The SPE interchange process

3. WEB SERVICES

A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network [W3C 2001]. Web Services are self-contained, self-describing, modular applications that can be published, located and invoked across the Web.

In order to access a Web Service, a client only needs to know this service definition and not how the service has been implemented. Therefore, clients and servers do not need to be written in the same language.

There are 3 main components in the Web Service environment, all of them XML (eXtensible Markup Language) based: SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery, and Integration).

SOAP is the standard protocol for accessing Web Services, making possible the communication between applications and their information interchange. It is an XML-based protocol for messaging and remote procedure calls that works on existing transport protocols, such as TCP, HTTP, SMTP, etc. A SOAP message is an XML document with a structure consisting of four basic parts: envelope, header, body and fault.

Copyright 2005 by the authors. All rights reserved.

WSDL is an XML-based language for describing Web Services and how to access them. It specifies the endpoint of the service, the operations that it offers, and the input and output of a Web Service.
Finally, UDDI is a registry where available services are published. Thanks to the registry, end users can easily find Web Services information.

We can now define a Performance model Web Service in terms of these technologies as: A Performance model Web Service is a performance modeling tool service exposed on the Web through SOAP, described with a WSDL file, and registered in UDDI.

4. PERFORMANCE MODEL INTERCHANGE FORMATS

Standard model interchange formats are the foundation that enables a Performance model Web Service. A common set of XML based interchange formats lets one use a variety of different tools as long as they support the format. Without them each tool would have to implement a custom interface to every other tool.

To use an interchange format, each tool must either provide an explicit import and export command, or provide an interface to/from a file. With a file interface, an Extensible Stylesheet Language translation (XSLT) [W3C 2001] can convert between the interchange format and the file. The translation can be relatively easy.

There are two performance model interchange formats as shown in Figure 1. The Performance Model Interchange Format (PMIF) is for the exchange of Queueing Network Models (QNM); and the Software Performance Model Interchange Format (S-PMIF) is for the exchange of software performance models among (UML-based) software design tools and software performance engineering tools. Each is described in the following paragraphs.

Earlier work defined a PMIF derived from a meta-model for system performance models that are based on QNMs [Smith and Williams 1999; Williams and Smith 1995]. The PMIF meta-model is a model of the information that goes into constructing a QNM model. The PMIF was subsequently enhanced, implemented in XML, and named PMIF 2.0 in [Smith and Lladó 2004].

Thus, the 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. 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.

Earlier work also defined an SPE meta-model that formally defines the information required to perform an SPE study [Williams and Smith 1995]. This model is known as the SPE meta-model because it is a model of the information that goes into constructing an SPE model.

The S-PMIF based on the SPE meta-model 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.

5. WEB SERVICE PROTOTYPE

To demonstrate the viability of a Performance model Web Service, we have developed a prototype in which the modeling tool that uses the Web Service is *SPE·ED*, and the tool that solves the model is Qnap.[*] Such development requires the implementation of the *SPE·ED* PMIF export mechanism and the Qnap PMIF import mechanism as well as the design and implementation of the Web Service itself.

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

The following sections discuss specific issues in exporting from *SPE·ED*, importing into Qnap and implementing the Web Service.

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

SPE·ED uses the Document Object Model (DOM) [W3C 2001] to export the pmif.xml. It creates the entire document in memory, and then writes it to a file.

[*] Qnap is a commercial tool developed by Simulog [Simulog] therefore, the Web Service described in this paper is only implemented as a prototype and for research purposes.

Copyright 2005 by the authors. All rights reserved.

Elements and attributes can be added in any order as long as they are in the correct location. It is a relatively small file, e.g., 2-3K for this case study, so the memory requirements are modest.

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

Several 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 [Smith and Lladó 2004].

5.2 Importing a pmif.xml model into Qnap

Qnap reads the input (QNM specification and solving parameters) from a file. Ultimately, Qnap would have an interface that would read from its standard file OR the pmif.xml file. However, we did not have access to Qnap source code and we could not implement such an interface directly. Therefore, we translated the pmif.xml file into a file in Qnap's format.

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.

5.3Web Service Design and Implementation

As stated earlier, we created a Web Service that solves performance models specified in PMIF format, in which the solving tool is Qnap. This requires the following steps:

- Validating the PMIF XML validation against the PMIF Schema www.perfeng.com/pmif/pmifschema.xsd
- Transforming a PMIF file into a file in Qnap's format.
- Executing Qnap with the transformed Qnap file and returning results

The execution function consists of two alternatives depending on the format in which the results have to be returned. The results can be sent back to the client as a Qnap output file (the client that uses the Web Service might actually know how to process the result file that Qnap normally produces). Additionally, the results can be returned as an XML file in a standard format.

A separate research project is focusing on the development of the meta-model for the results specification and on its XML Schema definition. In the meantime, this project uses a preliminary draft of this schema to demonstrate the feasibility of this approach. In this case, after the Qnap execution, the Qnap results file is transformed into a file that follows the XML results schema and it is sent back to the client.

The prototype XML results schema can be found in [Rosselló et al. 2005]. This schema includes the principal results for Workloads (response time and throughput), as well as the principal results for each Node (queue-server): throughput, response time, utilization, queue length, and service time - the overall total as well as the details for each Workload at the Node.

We chose the Apache HTTP Server [Apache 1999-2004] as the server for our prototype implementation since it is a widely used open-source HTTP server, efficient, secure and extensible. We also chose a widely used general-purpose scripting language, PHP, which is especially suited for Web development and can be embedded into HTML. The PHP Extension and Application Repository (PEAR) [PEAR-PHP 2001-2004] is a framework and distribution system for reusable PHP components, which are provided free of charge in the form of *packages*. One of these packages includes the SOAP protocol implementation and services. Other PHP functions handle XSLT and XML. The PEAR::SOAP library functions facilitate the server programming.

Copyright 2005 by the authors. All rights reserved.

The server functionality described above is implemented using the PHP language. It consists of the following four methods:

1. *XMLValidate*: validates the XML file against the PMIF XML-Schema
2. *QnapTransform*: receives a file in pmif.xml format and returns the result of the transformation of the PMIF document into Qnap's code (which is obtained by applying the Qnap importing mechanism explained in section 5.2),
3. *QnapExecute (Text results)*: returns the Qnap output execution results,
4. *QnapExecute (XML results)*: returns the results in XML format that follows the prototype version of the results schema.

The WSDL file, that describes these methods, uses the PEAR::SOAP extension. A sample of this WSDL file, which only deals with the validation function, is shown below.

```
<?xml version="1.0"?>
<definitions name="ExecucioQnapServer"
 targetNamespace="urn:ExecucioQnapServer"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="urn:ExecucioQnapServer"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
 <types xmlns="http://schemas.xmlsoap.org/wsdl/">
 </types>
 <portType name="ExecucioQnapServerPort">
  <operation name="validarXML">
   <input message="tns:validarXMLRequest" />
   <output message="tns:validarXMLResponse" />
  </operation>
 </portType>
 <binding name="ExecucioQnapServerBinding"
          type="tns:ExecucioQnapServerPort">
  <soap:binding style="rpc"

transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="validarXML">
  <soap:operation soapAction="urn:
ExecucioQnapServer#ExecucioQnapServer#validarX
ML"/>
   <input>
    <soap:body use="encoded"
    namespace="urn:ExecucioQnapServer"
    encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   <output>
    <soap:body use="encoded"
    namespace="urn:ExecucioQnapServer"
    encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"/>
   </output>
  </operation>
 </binding>
 <message name="validarXMLRequest">
  <part name="inputString" type="xsd:string"/>
 </message>
 <message name="validarXMLResponse">
  <part name="outputString" type="xsd:string"/>
 </message>
</definitions>
```

Additional details of this prototype and sample code can be found in [Rosselló et al. 2005].

6. EXPERIMENTAL RESULTS

In order to access a specific Web Service, the *end user* needs first to query the Web Service WSDL file that shows the description of the methods and their parameters, and then to develop a client that accesses one or more of these methods.

The user is free to create his own client and implement it with any technology that provides the standard SOAP. In the following paragraphs we describe two client implementations that we have developed to show different possibilities for the *end user* and to prove that our Web Service implementation works independently of the client platform. Both clients carry out the same tasks: connection to the Web Service server and call to all the methods previously described.

One of these client implementations is based on Apache Server, the PHP language and the PEAR::SOAP module. Therefore, in this case, the client needs to have Apache installed. This could be the case when it is an application server that wants to access a Web Service on another machine. This client can be executed from any Web browser and its user interface could be like the one shown in Figure 2. Each of the buttons in the figure invokes the Web Service call to one of the four methods described in the previous section.

The second client implementation is written in VBScript. With this client a Windows XP user does not need to install Apache or any other extra software since XP has a SOAP client already installed.

Copyright 2005 by the authors. All rights reserved.

To create the Web Service client you need to:

- Execute a WSDL file query
- Choose the desired method to invoke and its parameters and types.
- Invoke the URL where the server programmed methods reside with the required parameters.

The following shows an example of the use of the procedure for the PHP implementation. The example we use is the ATM example from [Smith and Lladó 2004; Smith and Williams 2002]. It is an open model with 2 workload classes.

First, the software model was created in the *SPE·ED* performance modeling tool. It was then exported to the PMIF 2.0 format.

Next, the pmif.xml file is copied into the *XML to send* screen area (shown in Figure 2). Then the user clicks on one of the five buttons in the figure and the corresponding Web Service function is invoked by the client.

If for instance, the button pressed is the *Execute (Text results)*, the Web Service validates the file (successfully), and then translates it to Qnap input using the XSLT translation. The model is solved and the results are returned in the Qnap output file format.

The *Execute (XML results)* button demonstrates the feasibility of the XML results schema approach. The values from the XML results file for this case study are summarized in Table 1. Notice that *SPE·ED* does not solve multi-class models analytically. The analytic results from Qnap (in row 2) 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 Web Service with the results schema was successful.

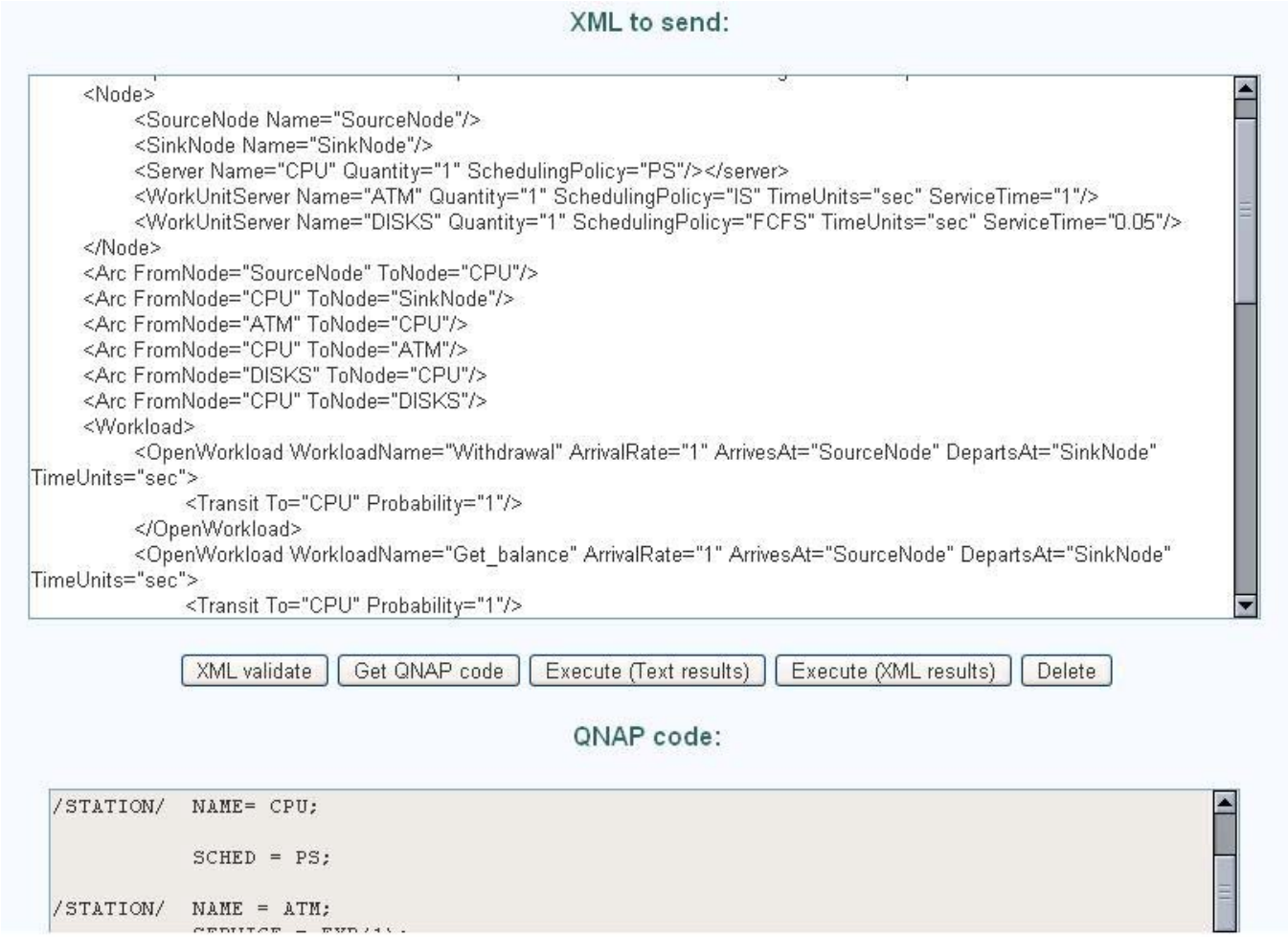

Figure 2. PHP client: user's interface.

Copyright 2005 by the authors. All rights reserved.

Copyright 2005 by the authors. All rights reserved.

Table 1: Model results for the ATM model study (A: Analytical results, S: Simulation results)

Model Study	Response Time		CPU Utilization		Disk Utilization		Confidence/SimTime	
	SPE·ED	Qnap	*SPE·ED*	Qnap	*SPE·ED*	Qnap	*SPE·ED*	Qnap
1.ATM (S)							.314/49890	95\%/5000
Withdrawal	11.971	11.9	0.006	0.0063	0.403	0.3984		0
GetBalance	6.354	6.362	0.003	0.0025	0.151	0.1519		
2.ATM (A)								
Withdrawal		11.9		0.0063		0.4		
GetBalance		6.336		0.0025		0.15		

7. CONCLUSIONS

This paper described some valuable uses for Performance model Web Services that enable "plug and play" use of various performance modeling tools. It then showed some ways they might be used for Software Performance Engineering to automate the evaluation of software architectures and designs. The "plug and play" concept for SPE has the potential to dramatically improve the automation of SPE tasks. Further information on this approach is in [Smith et al. 2005]. The paper then reviewed information about Web Services and the technology used to implement them. Next it described previous work on two XML based performance model interchange formats that provide a foundation for interchanging performance models among tools. The PMIF is for exchanging Queueing Network Models, and the S-PMIF is for exchanging software performance models. Next we described the implementation of our prototype Performance model Web Service. Finally we gave some experimental results that show the viability of Performance model Web Services.

This work is part of an ongoing project to automate SPE performance evaluation. The interchange formats that were previously developed allow flexibility in when and how performance specifications are provided and even allow some specifications to be provided by measurement tools.

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 and Web Service when tool vendors do not provide a custom interface.

The interchange also enables a "plug and play" paradigm for using performance modeling tools appropriate for the particular problem to be studied. These interchange formats have established a foundation that makes implementation of a Performance model Web Service viable.

Additional work is underway to automate the call to the Web Service from *SPE·ED,* to parse the XML results, and to display them graphically in the software performance model. A separate research project is also developing the complete meta-model for results and the XML schema for it. There are some additional issues to be investigated associated with the use of proprietary tools as Web Services, such as authenticating users and perhaps charging for services.

As previously mentioned, this prototype Web Service validates an input file against the PMIF 2.0 XML schema [Smith and Lladó 2004]. Therefore, the input model is checked for possible syntactical inconsistencies. However, the PMIF XML schema does not handle semantic validation (for example, that declared nodes are actually used in the model, the model topology is valid, etc.). We did not include semantic validation because it is reasonable to assume that production tools generate correct pmif.xml, and that it is only necessary to validate the semantics occasionally. Semantic validation would be useful for a PMIF that has not been automatically generated, such as when an *end user* writes a PMIF model to be sent to the Web Service. A PMIF semantic validation would also be useful as a Web Service.

ACKNOWLEDGEMENTS

The authors would like to thank Jeronia Rosselló for implementing the prototype, and the ACSIC research group at the *Universitat de les Illes Balears* for the help offered towards this collaboration.

Copyright 2005 by the authors. All rights reserved.

8. REFERENCES

[Apache 1999-2004] Apache, "Apache http server project", http://www.apache.org, 1999-2004.

[L&S] L&S, Computer Technology, Inc., Performance Engineering Services Division. # 110, PO Box 9802. Austin, TX 78766, (505) 988-3811, www.perfeng.com.

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

[PEAR-PHP 2001-2004] PEAR-PHP, "The php extension and application repository", http://pear.php.net, 2001-2004.

[Rosselló et al. 2005] Rosselló, J., C. M. Lladó, R. Puigjaner and C. U. Smith, "A Web Service for Solving Queueing Network Models Using PMIF", www.perfeng.com/paperndx.htm, April 2005.

[Simulog] Simulog, Paris Office. 1 rue James Joule, 78286 Guyancourt Cedex. France, 33 (0)1 30 12 27 00, www.simulog.fr.

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

[Smith et al. 2005] Smith, C. U., C. M. Lladó, V. Cortellessa, A. Di Marco and L. G. Williams, "From UML Models to Software Performance Results: An SPE Process Based on XML Interchange Formats", Workshop on Software and Performance (WOSP) 2005, Palma de Mallorca, ACM, 2005.

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

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

[W3C 2001] W3C, "World Wide Web Consortium", www.w3c.org, 2001.

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