Impact of SOAP Implementations in the Performance of a Web Service-Based Application*

Elena Gómez-Martínez and José Merseguer

Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza
C/María de Luna,1 50018 Zaragoza, Spain
{megomez, jmerse}@unizar.es

Abstract. This article recalls, from the literature, a performance study of a web service. That study, based on the layered queuing network (LQN) paradigm, is now addressed following the PUMA approach to obtain a new performance model, in this case in terms of Petri nets, for the target web service. Such Petri net model is used to extend the previous LQN results with respect to some key web service performance aspects: the SOAP toolkit and the XML parsers. Actually, this paper aims to explore through a case study some of the main concerns of web services performance at the middleware layer. The acquired background is meant to start to develop a methodology, based on the SPE principles, useful to analyze web services performance.

1 Introduction

A web service is a collection of protocols and standards used for exchanging of XML messages between applications. Unlike other middleware technologies [2], they allow to communicate heterogeneous environments deployed on the network, offering flexibility and interoperability.

Performance is one of the key aspects and probably the Achilles' heel of web services and in general of services offered over the Internet [28]. However, it has not been adequately addressed from a formal modeling viewpoint in the literature yet. In this work, we try to overcome some aspects of this lack by accomplishing an in-depth study of different key aspects of web services performance at the middleware layer: the SOAP implementations and the XML parsers.

This work is proposed as a first step to develop a methodology to evaluate web service performance, and we start addressing some middleware performance issues. The methodology will use Petri nets (PN) [1] as formal method, and will follow the Software Performance Engineering (SPE) [19] principles and the Performance by Unified Model Analysis (PUMA) approach [30]. PUMA aims translations from different kinds of design models and performance models.

* This work was supported by the projects TIC2003-05226 and DPI2006-15390 of the Spanish Ministry of Science and IBE2005-TEC-10 of the University of Zaragoza.

Our study is based on an interesting performance study of a web service developed, also under the SPE principles, in [3]. We rearchitect this case study following the PUMA approach to get a Generalized Stochastic Petri Net (GSPN) [1]. The GSPN, properly analyzed with the TimeNET tool [23], allows us to offer interesting results about performance middleware key aspects and to contrast them with the results obtained from pragmatic (non-formal) studies.

The rest of the paper is organized as follows. Section 2 revises the state of the art and places our proposal for study performance of web services in the current scene. Section 3 addresses key issues concerning performance of web services at middleware layer. Section 4 recalls the web service under study and we obtain the PN that models the target system. Such net will be useful, in section 5, to accomplish the study proposed in this work. Therefore, the impact of SOAP implementations and XML parsers is studied by means of that formal model. The article ends in section 6 giving the conclusion.

2 Related Work

Performance is an important aspect of web services. Nevertheless, from the best of our knowledge, very few papers focuss on performance evaluation of web service-based applications. And a very few of them follow the techniques proposed by the SPE [19].

Menascé and Almeida [13] developed a methodology from we have learnt the key issues of performance evaluation of web services. While this methodology is focussed on capacity planning using queuing networks (QN), we aim at its performance prediction using PN and the Unified Modeling Language (UML).

Chandrasekaran et al. [4] propose a simulation technique for analyzing performance of composite web services in order to obtain efficient web processes. Menascé in [12] studies QoS issues of composite web services. In [26], Datla and Goševa-Popstojanova present a measurement-based study of performance of e-commerce applications. They study the impact of web services together with other components on integrated applications using benchmark techniques. Ng et al. [14] evaluate diverse SOAP implementations by means of benchmarks of a simple service with three types of message. In contrast to us, they probe that serialization and deserialization are the primary important bottleneck for this application.

Liu et al. [11] propose an approach to predict performance metrics for a middleware-hosted application using QN models. Although, this work is focussed on a J2EE application, their modeling approach is suitable to other middleware technologies, such as CORBA and COM+/.NET.

Verdickt et al. [27] propose a Model Driven Architecture (MDA) model transformation for Platform Specific Models (PSM), including middleware performance details. It is based on SPE and the UML-SPT [15] profile. The transformation process is made by a tool which generates LQN models.

Gilmore et al. [9] propose an UML-based methodology for analyzing security and performance aspects using PEPA models. This method is implemented in the Choreographer design platform.

3 Performance Issues of the Web Services

Web service technology has not been developed with performance as a goal. Performance issues affect several aspects: the XML protocols, such as discovering using UDDI [13], transporting using usually HTTP [7], the latency of SOAP implementations [5] or the use of an XML parser [10]. Furthermore, web services can be provided with dynamic composition of web services, affecting performance in any way [4,12]. The software infrastructure is other significant factor [11].

Although all of these issues are relevant, this paper focusses on those that being closer to the middleware layer can be parameterized in a UML design. Among them, SOAP implementation is one of the key factors that have influence on performance, as previous studies have shown [5,7,10]. Therefore, it is important to determine which particular SOAP toolkit can meet the performance requirements of an application. These studies remark the following topics:

Serialization is the process to convert an in-memory object into an XML stream. This includes to pack the XML message in the SOAP envelope and to build the message which will be sent by the corresponding transport protocol, mainly HTTP [10].

Deserialization converts XML streams in wire-format objects in memory. In this process two phases must be emphasized: (1) unpacking the SOAP envelope and (2) parsing and interpreting the XML document. The most widely models used for parsing are *Document Object Model* (DOM) [6], *Simple API for XML* (SAX) [18] and *XML Pull Parser* (XPP) [8]. DOM parsers are suitable for small documents which must be validated and/or modified. SAX parsers are better for large documents. XPP is optimized when the XML elements are processed in succession and do not need to be visited again. The parser process has a great impact in the performance of SOAP implementation, as previous works have studied [7,10]. Note that XML native parsers and those embedded in SOAP have to be differentiated, since they exhibit different features and performance characteristics [21,8].

However, not only these processes affect the performance of a SOAP toolkit, others such as data structure support, optimizations to handle scientific data or algorithms implementation and protocols have influence too [5,10]. These topics will not be addressed in this work, since they are out of scope of the case study which guides it. Other significant factors that may impact in performance are the service processing time, i.e. the *business logic*, and the XML file size [3].

The goal of this paper is to study the impact of the following aspects in web service performance: (**G1**) XML parsers and (**G2**) SOAP toolkits. Furthermore, other factors that may impact in performance will be studied, such as (**G3**) the sensibility of a web service with respect to the document file size exchanged and (**G4**) the service processing time. The implementations of XML parsers under consideration are: Xerces [24], Xerces2 [25], Crimson [22] and XML Pull Parser (XPP) from [8]. The SOAP toolkits considered are AxisJava and .NET, since they are widely used. We have also included XSUL for its excellent performance for large documents [8].

4 SPE for Web Services

In order to study the impact of the previous goals, we recall a performance case study taken from [3], then section 4.1 summarizes it and its results. In section 4.2 we apply the PUMA approach to obtain a GSPN model from the UML system description.

4.1 Case Study: CDSS Web Service

Catley et al. propose in [3] *"an infrastructure to support artificial intelligence-based clinical decision systems (CDSSs). The system processes multidomain medical data in high-risk medical environments in order to reduce medical errors and alert detection systems"*. It integrates and accesses CDSSs and distributed databases from different medical domains in order to predict medical outcomes. These CDSSs are offered as web-services. The paper models a representative subset of this infrastructure, which invokes a CDSS as a web service and accesses the patient's Electronic Patient Record (EPR). Figure 2(a) depicts the sequence diagram (SD) of such CDSS invocation process, that proposes an *initial system configuration* made of one instance per hardware and software resource.

The system parses XML documents using the Xerces parser through a DOM interface. The *required* response time for 50 users requesting the system should not exceed 8 seconds.

Catley et al. applied the SPE techniques developed in [16] to assess the required metric. Then they modeled the system by means of deployment and sequence diagrams annotated according to the *UML Profile for Schedulability, Performance and Time Specification* (UML-SPT) and translated them into an LQN model [29].

This model was solved with the *initial configuration*, determining that the system can not meet the performance target, see Figure 1(a1) where the response time for 50 users is 39.9 seconds. They identified system bottlenecks and proposed a *new configuration* that replicates processors and threads (10 `WSCoordinator`, 10 `CDSS`, 3 `AppCPU` and a variable number of `EPR`). Figure 1(a2) depicts the results of multithreading the `EPR` task when the system is executed by 50 users. They determined that the target is achieved in this new configuration with 8 threads of `EPR`.

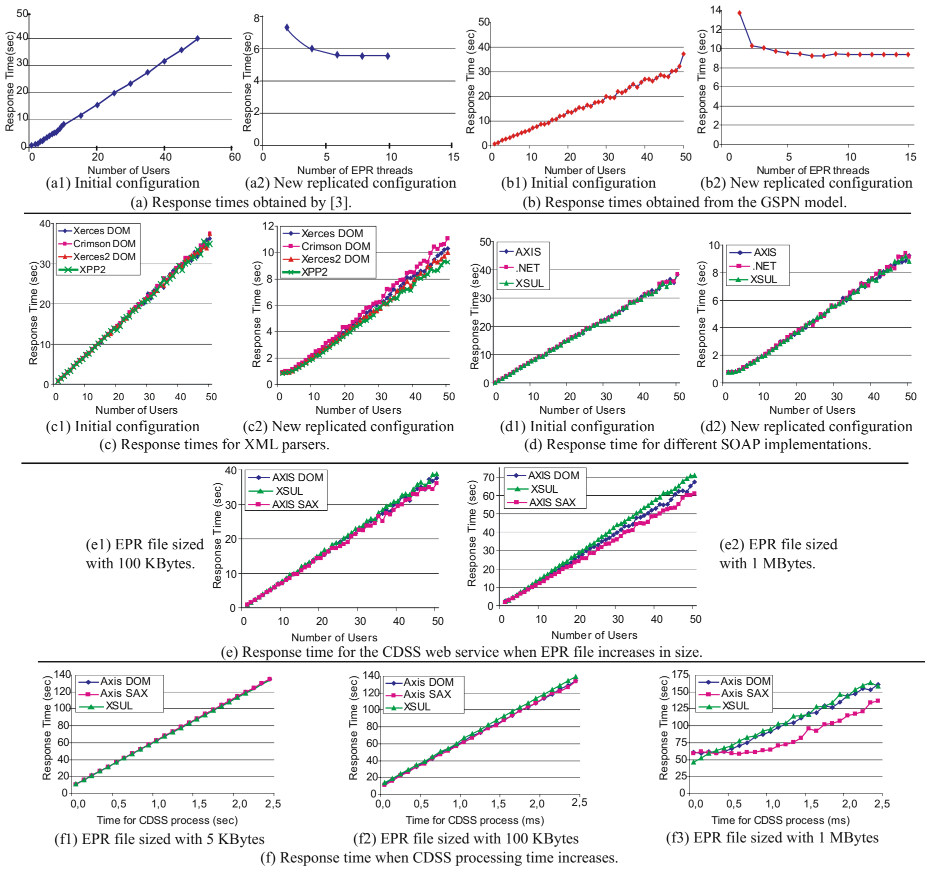
4.2 Applying the PUMA Approach

PUMA [30] was designed as a framework to obtain performance models from design models. Therefore, we use PUMA to obtain a GSPN model from the SD in Figure 2(a), which models the CDSS. The GSPN model aims for validating the CDSS results in [3] and for dealing with the performance goals previously given.

PUMA uses an intermediate model, the *Core Scenario Model* (CSM) [17], which is suited to produce a performance model, such as layered and regular QNs, and stochastic PNs.

The CSM defines a performance *Scenario* as a sequence of *Steps* that are linked by *Connectors*. A Step is a sequential piece of execution. Connectors can include branches, merges, and forks and joins. A scenario has a *Start* point and an *End* point. Start points are associated with *Workload*. There exist two kind of *Resources*: *Active*, which execute steps, and *Passive*, which are acquired and released during scenarios by special *ResAcquire* and *ResRelease* steps. Steps are executed by (software) *Components* which are passive resources.

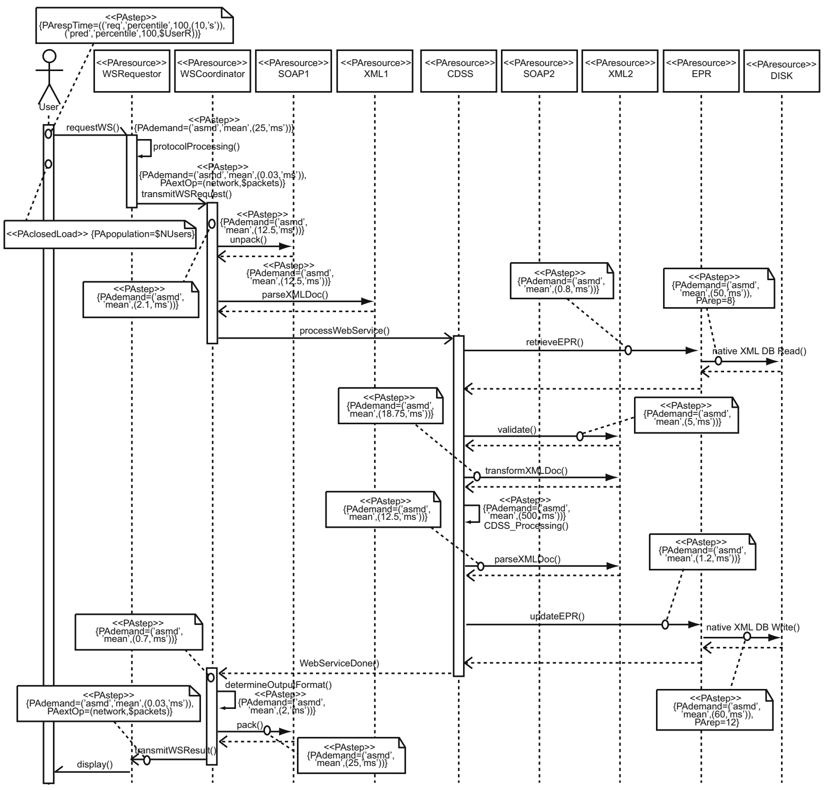
PUMA gives a translation process to get a CSM model from a UML SD. Figure 3(a) depicts the resulting CSM for our target SD. Observe that this CSM is made of two scenarios, the one corresponding to the CDSS invocation process (left column) and the CDSS processing (right column). The CDSS processing scenario comprises the messages from `processWebService()` to `WebServiceDone()`. The other messages of the SD correspond to the CDSS invocation process.



(a1) Initial configuration (a2) New replicated configuration
(a) Response times obtained by [3].
(b1) Initial configuration (b2) New replicated configuration
(b) Response times obtained from the GSPN model.
(c1) Initial configuration (c2) New replicated configuration
(c) Response times for XML parsers.
(d1) Initial configuration (d2) New replicated configuration
(d) Response time for different SOAP implementations.
(e1) EPR file sized with 100 KBytes.
(e2) EPR file sized with 1 MBytes.
(e) Response time for the CDSS web service when EPR file increases in size.
(f1) EPR file sized with 5 KBytes (f2) EPR file sized with 100 KBytes (f3) EPR file sized with 1 MBytes
(f) Response time when CDSS processing time increases.

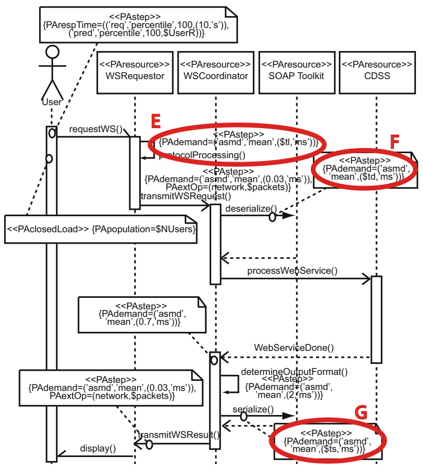
Fig. 1. Results of the experiments

The CSM in Figure 3(a) is translated into a GSPN, see Figure 3(b), by means of a extraction process developed in [30]. So, each class of the CSM corresponds



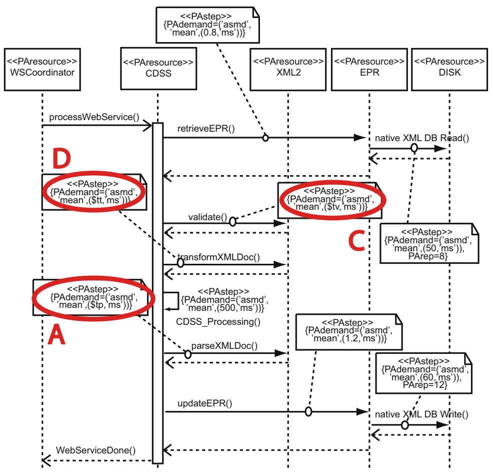
<<PAstep>> {PArespTime=(('req','percentile',100,(10,'s')), ('pred','percentile',100,$UserR))}
User
<<PAresource>> WSRequestor
<<PAresource>> WSCoordinator
<<PAresource>> SOAP1
<<PAresource>> XML1
<<PAresource>> CDSS
<<PAresource>> SOAP2
<<PAresource>> XML2
<<PAresource>> EPR
<<PAresource>> DISK
requestWS()
<<PAstep>> {PAdemand=('asmd','mean',(25,'ms'))}
protocolProcessing()
<<PAstep>> {PAdemand=('asmd','mean',(0.03,'ms')), PAextOp=(network,$packets)}
transmitWSRequest()
<<PAclosedLoad>> {PApopulation=$NUsers}
<<PAstep>> {PAdemand=('asmd', 'mean',(12.5,'ms'))}
unpack()
<<PAstep>> {PAdemand=('asmd', 'mean',(12.5,'ms'))}
parseXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(2.1,'ms'))}
processWebService()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.8,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(50,'ms')), PArep=8}
retrieveEPR()
native XML DB Read()
<<PAstep>> {PAdemand=('asmd', 'mean',(18.75,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(5,'ms'))}
validate()
transformXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(12.5,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(500,'ms'))}
CDSS_Processing()
<<PAstep>> {PAdemand=('asmd', 'mean',(1.2,'ms'))}
parseXMLDoc()
updateEPR()
native XML DB Write()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.7,'ms'))}
WebServiceDone()
determineOutputFormat()
<<PAstep>> {PAdemand=('asmd', 'mean',(2,'ms'))}
<<PAstep>> {PAdemand=('asmd','mean',(0.03,'ms')), PAextOp=(network,$packets)}
<<PAstep>> {PAdemand=('asmd', 'mean',(60,'ms')), PArep=12}
pack()
transmitWSResult()
<<PAstep>> {PAdemand=('asmd', 'mean',(25,'ms'))}
display()

(a) Sequence diagram taken from [3] describing the system.



<<PAstep>> {PArespTime=(('req','percentile',100,(10,'s')), ('pred','percentile',100,$UserR))}
User
<<PAresource>> WSRequestor
<<PAresource>> WSCoordinator
<<PAresource>> SOAP Toolkit
<<PAresource>> CDSS
requestWS()
E
<<PAstep>> {PAdemand=('asmd','mean',($tl,'ms'))}
protocolProcessing()
F
<<PAstep>> {PAdemand=('asmd', 'mean',($td,'ms'))}
<<PAstep>> {PAdemand=('asmd','mean',(0.03,'ms')), PAextOp=(network,$packets)}
transmitWSRequest()
deserialize()
<<PAclosedLoad>> {PApopulation=$NUsers}
processWebService()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.7,'ms'))}
WebServiceDone()
determineOutputFormat()
<<PAstep>> {PAdemand=('asmd', 'mean',(2,'ms'))}
<<PAstep>> {PAdemand=('asmd','mean',(0.03,'ms')), PAextOp=(network,$packets)}
serialize()
G
<<PAstep>> {PAdemand=('asmd', 'mean',($ts,'ms'))}
transmitWSResult()
display()

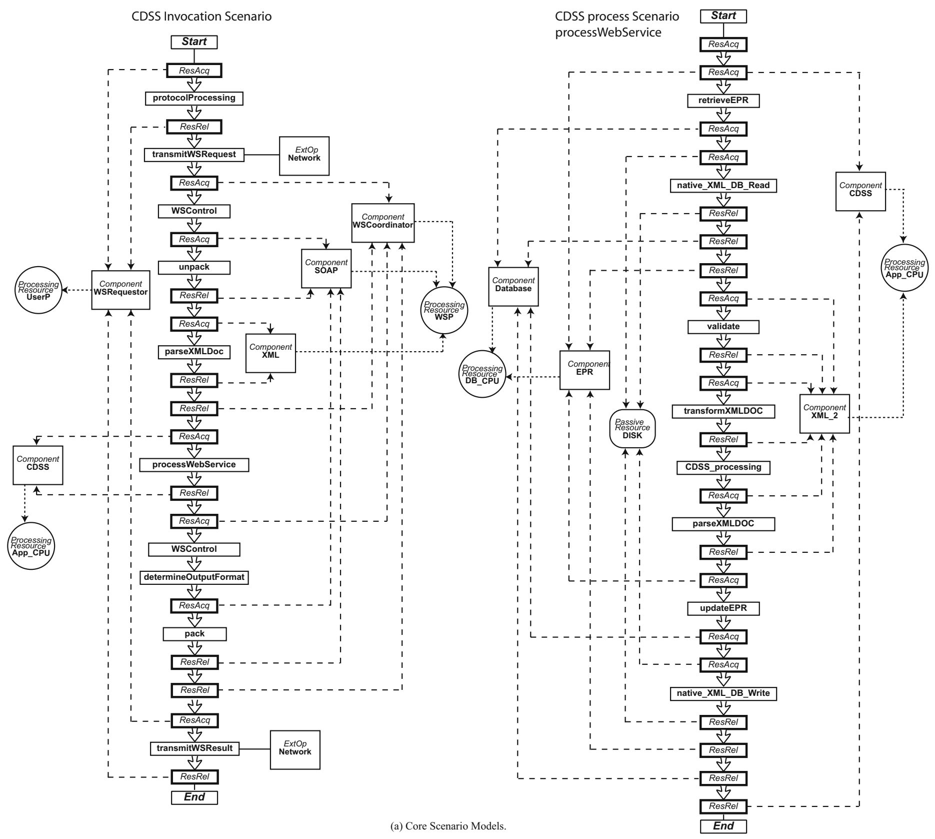
(b) SD describing the proposed key performance scenario with SOAP toolkit.



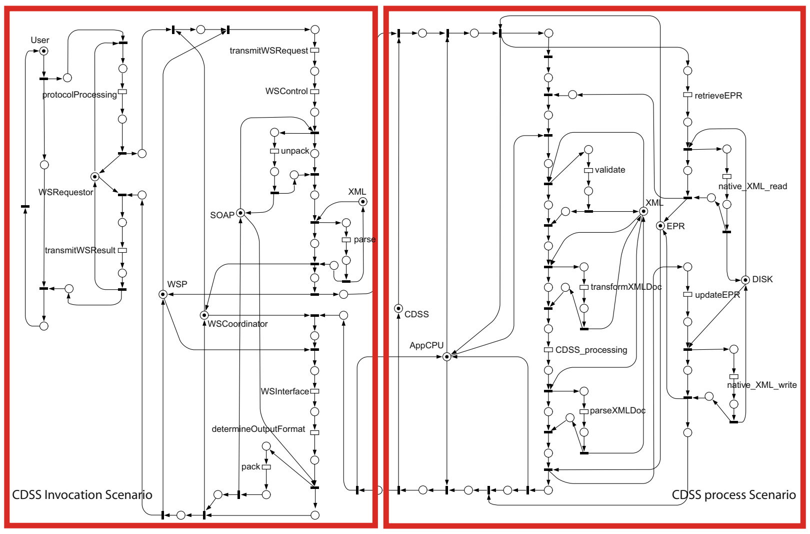
<<PAstep>> {PAdemand=('asmd', 'mean',(0.8,'ms'))}
<<PAresource>> WSCoordinator
<<PAresource>> CDSS
<<PAresource>> XML2
<<PAresource>> EPR
<<PAresource>> DISK
processWebService()
retrieveEPR()
native XML DB Read()
D
<<PAstep>> {PAdemand=('asmd', 'mean',($tt,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',($tv,'ms'))}
validate()
C
<<PAstep>> {PAdemand=('asmd', 'mean',(50,'ms')), PArep=8}
transformXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',($tp,'ms'))}
A
<<PAstep>> {PAdemand=('asmd', 'mean',(500,'ms'))}
CDSS_Processing()
<<PAstep>> {PAdemand=('asmd', 'mean',(1.2,'ms'))}
parseXMLDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(60,'ms')), PArep=12}
updateEPR()
native XML DB Write()
WebServiceDone()

(c) Changes in the CDSS w.r.t. the original proposal.

Fig. 2. Sequence diagrams



(a) Core Scenario Models.



(b) GSPN representing the SD in Figure 2 (a).

Fig. 3. Core Scenario Model and LGSPN for the SD in Figure 2(a)

with a GSPN pattern. For instance, a step is translated into a timed transition with an input place, where its delay is the demand attribute of the step. All of the GSPN patterns are composed until the GSPN representing the whole scenario is built.

Performance metrics can be obtained using TimeNET [23] to solve this GSPN by means of simulation techniques. Figure 1(b1) and Figure 1(b2) present the same experiments as Figure 1(a1) and Figure 1(a2), respectively, i.e. the response times for the initial and the new replicated configuration. In Figures 1(b2) and 1(a2), the response time for 50 users is stated in 9.23 seconds and 5.4 seconds, respectively. These response times are greater since PNs introduce synchronization in the model. However, both the order of magnitude and the tendency of the results are kept.

Once it has been verified that the results, obtained by the derived GSPN, are similar to those obtained in [3] with LQN, the following step is to study, using this GSPN, the impact of XML parsers and SOAP implementations.

5 Web Services Tunning: CDSS Performance Improvements

In this section we exploit the CDSS case study to deal with the goals **(G1)**, **(G2)**, **(G3)** and **(G4)** proposed in section 3. The final objective is to extract conclusions about those key aspects of web service performance from the case study.

5.1 Impact of XML Parsers

Since XML parsers affect web service performance [7,5,10], we explore different alternatives of them in order to study their impact in the CDSS web service.

We realized that some of the CDSS parameters in [3] should be changed for the following considerations:

A *Document build time* is the time to scan and interpret the XML document [20], but in [3] is assigned to the packing operation. In our experiments, we will assign this value for parsing operations, see Table 1.

B *Document modify time* is the time required to systematically modify the constructed document representation [20], but in [3] is assigned to the parsing operations. We do not assign this value to an operation, since we consider that EPR file is not updated.

C *Document walk time* is the time required to walk the constructed document representation [20]. As [3], we will assign it to validate the XML document, see Table 1.

D *Text generation time* is the time required to output document representations as text XML documents [20]. As [3], we will assign it to transform the XML document, see Table 1.

Table 1. Performance parameters for XML operations from [21]

Operation	parameter in SD	Mean Execution Time (ms) Xerces	Xerces2	Crimson	XPP
(A) `parseXMLDoc()`	\$tp	6.957	2.898	9.856	1.159
(C) `validate()`	\$tv	$\simeq 0$	$\simeq 0$	$\simeq 0$	$\simeq 0$
(D) `transformXMLDoc()`	\$tt	1.055	1.231	1.231	0.703

Table 1 gives the new values taken from an updated benchmark [21]. Figure 2(c) depicts the part of the SD that has been changed to consider the new values in the model.

Figure 1(c1) and Figure 1(c2) depict the response times when the parameters in Table 1 are applied. These results can be compared with those in Figure 1(a1) and Figure 1(a2), as well as with those in Figure 1(b1) and Figure 1(b2), being similar in all cases. Therefore, it does not matter which parser is used.

However, according to [7,21], the response times for Xerces parsers are worse than the obtained ones for Crimson or XPP parsers. Slightly best results are obtained by XPP. The reason for our results is the small size of the EPR file, only 5 KBytes. So, in this case, the XML parser significantly does not affect the performance of the CDSS web service. But in section 5.3 we try to validate the conclusions in [7,21] by varying the EPR file size.

5.2 Impact of SOAP Implementations

Currently, several implementations of SOAP are emerging and their performance differs to a great extent [5,10]. Therefore, it is profitable to determinate what toolkit meets performance objectives in the CDSS invocation web service.

We guess that in [3] the SOAP parameters are taken from [20], but we consider more appropriate to use an specific SOAP benchmark, taken from [10]. Table 2 provides the values of SOAP operations, (F) deserialization and (G) serialization and the overhead that the SOAP toolkit imposes, (E) the latency. Note that they have been calculated assuming that most of the content of the EPR file are strings. Figure 2(b) depicts the part of the SD changed to include in the CDSS these new parameters.

Figure 1(d1) shows that all the SOAP toolkit give similar response time for the CDSS. Only XSUL performs a little better in the replicated configuration, see Figure 1(d2). Comparing these results with those in Figure 1(a1), they are alike. We guess that as the SOAP message, which contains the EPR file, is small, the time taken by processing SOAP is negligible with respect to the CDSS

Table 2. Performance parameters for SOAP toolkits from [10]

Operation	parameter in SD	Mean Execution Time (ms) AxisJava	.NET	XSUL
(E) Latency → `protocolProcessing()`	\$tl	8.35	3.5	2.435
(F) Deserialization → `deserialize()`	\$td	10.476	4.797	3.935
(G) Serialization → `serialize()`	\$ts	16.151	4.481	3.706

processing time. In section 5.4, we try to verify this affirmation by varying this service processing time.

5.3 Impact of the EPR File Size

The previous experiments showed that due to the small size of the EPR file, both the XML parser and the SOAP implementations have no relevant impact for the performance of the CDSS web service.

Table 3. Performance parameters from [21] and [10]

Parameter	Mean Execution Time (ms)					
	100 KBytes			1 MBytes		
in SD	AxisJava DOM	AxisJava SAX	XSUL	AxisJava DOM	AxisJava SAX	XSUL
(A) \$tp	27.68	10.19	40.79	297.8	78.37	501.56
(C) \$tv	1.37	≃0	0.68	31.34	≃0	15.67
(D) \$tt	11.36	≃0	26.51	282.13	≃0	203.76
(E) \$tl	8.35	8.35	2.435	8.35	8.35	2.435
(F) \$td	44.39	44.39	26.78	917.11	917.11	431.464
(G) \$ts	32.00	32.00	35.53	291.82	291.82	265.81

If it would be considered that this file increases in size, the results could be different. Note that the XML-based EPR file is also enveloped in the SOAP message, therefore its size affects both XML and SOAP operations. Table 3 provides the new parameters, considering two sizes for the EPR, they are set in sequence diagrams of Figure 2(b) and Figure 2(c). We have taken into account that XPP is the native parser for XSUL and for Xerces through DOM or SAX interface for AxisJava.

Figures 1(e1) and 1(e2) show the response time with the initial configuration when the EPR file size is 100 KBytes and 1 MBytes, respectively. If we observe the results when EPR file size is 100 KBytes, these are similar to those when it is only 5 KBytes, see Figure 1(d1). However, the response time increases meaningfully with 1 MBytes. As expected [7], comparing the SOAP implementations, AxisJava through SAX interface outperforms AxisJava through DOM. Surprisingly, in spite of the good results of XSUL presented in [10] for large sizes, it performs poorly in this case. It may be due to the time required by XPP to build the document in memory, as suggested in [21].

5.4 Impact of the CDSS Processing Time

Once studied the impact of EPR file size, we come back to section 5.2 to verify if the time required for processing this EPR file (with SOAP and the XML parser) is irrelevant with respect to the time taken by the `CDSS_Processing()` process. In order to validate this supposition, the service processing time will be modified. See the annotation of the `CDSS_Processing()` message self dispatched by the CDSS in the sequence diagram depicted in Figure 2(a).

In section 5.1 and section 5.2, we guess that XML parser and SOAP toolkits have not influence in CDSS web service, since XML-based EPR file size is small.

Therefore, the time required for being processed it by SOAP and XML parser is irrelevant with respect to the time taken by `CDSS_Processing()` process. In order to validate this supposition, the service processing time is modified.

Figure 1(f1) depicts the response times for 50 users with the initial configuration when the `CDSS_Processing()` service time varies from 0.1 to 2.5 seconds and the EPR file is 5 KBytes; in Figure 1(f2) the EPR file is 100 KBytes and in Figure 1(f3), 1 MBytes. The response time of the different SOAP implementations and XML parsers follows the same tendency while sizing EPR file to "*small sizes*", 5 KBytes or 100 KBytes. However, when EPR file is 1 MBytes, "*big sizes*", and `CDSS_Processing()` is less than 0.4 seconds, AxisJava through SAX parser performs poorly compared to AxisJava through DOM and XSUL. But, when service time increases, AxisJava through SAX parser outperforms them. We guess that XSUL performs better when the service time is small because it is oriented to slightly processed scientific data. Similarly, we guess that DOM and SAX outperform XSUL when the processing time is greater than 0.4 seconds since they are conceived to process generic information which may be repeatedly accessed.

This experiment shows that the CDSS processing time (`CDSS_Processing()`) and the EPR file size condition the impact of the XML parser and the SOAP implementation.

6 Conclusion

This paper studies some of the main concerns of web services performance through a set of goals established on a CDSS web service. We have focussed on how XML parsers, SOAP implementations, the exchanged file size and the service time influence the web services.

Our experiments indicate that the XML parser choice slightly affects web services performance when the XML-based file size is small, whereas the SOAP implementation influence is even smaller. However, when the EPR file increases in size, the response times obtained are worst and there exist noticeable differences among XML parsers and SOAP implementations. These differences intensify when the service processing time changes.

We can conclude that the impact of the XML parsers and the SOAP implementations is conditioned by both XML-based file size and service time, i.e. the serialization and deserialization processes are not bottlenecks for large data applications and large service times.

Acknowledgments. The authors would like to thank Diego Rodríguez for his help in computing results using the TimeNET tool.

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley Series in Parallel Computing - Chichester, 1995.

2. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications.* Springer, 2004.
3. C. Catley, D. Petriu, and M. Frize. Software Performance Engineering of a Web service-based Clinical Decision Support infrastructure. In *ACM WOSP*, pages 130–138, 2004.
4. S. Chandrasekaran, J. Miller, G. Silver, I. Arpinar, and A. Sheth. Performance Analysis and Simulation of Composite Web Services. *Electronic Markets*, 13(2), 2003.
5. D. Davis and M. Parashar. Latency Performance of SOAP Implementations. In *IEEE CCGRID*, pages 407–412, 2002.
6. Document Object Model (DOM). `http://www.w3.org/DOM/`.
7. R. Elfwing, U. Paulsson, and L. Lundberg. Performance of SOAP in Web Service Environment Compared to CORBA. In *IEEE APSEC*, pages 84–96, 2002.
8. Extreme! Computing Lab. Indiana University. `http://www.extreme.indiana.edu/xgws/xsoap/xpp/`.
9. S. Gilmore, V. Haenel, L. Kloul, and M. Maidl. Choreographing Security and Performance Analysis for Web Services. In *EPEW/WS-FM*, pages 200–214, 2005.
10. M. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. van Engelen, K. Chiu, and M. Lewis. A Benchmark Suite for SOAP-based Communication in Grid Web Services. In *IEEE SC*, page 19, 2005.
11. Y. Liu, A. Fekete, and I. Gorton. Predicting the performance of middleware-based applications at the design level. In *ACM WOSP*, pages 166–170, 2004.
12. D. Menascé. Composing Web Services: A QoS View. *IEEE Internet Computing*, 8(6):88–90, 2004.
13. D. Menascé and V. F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
14. A. Ng, S. Chen, and P. Greenfield. An Evaluation of Contemporary Commercial SOAP Implementations. In *AWSA*, pages 64–71, 2004.
15. Object Management Group, `http://www.uml.org`. *UML Profile for Schedulabibity, Performance and Time Specification.*, 2005.
16. D. Petriu and H. Shen. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In *TOOLS*, volume 2324 of *LNCS*, pages 159–177. Springer, 2002.
17. D. Petriu and C. Woodside. A Metamodel for Generating Performance Models from UML Designs. In *UML*, volume 3273 of *LNCS*, pages 41–53. Springer, 2004.
18. Simple API for XML (SAX). `http://www.saxproject.org/`.
19. C. Smith and L. Williams. *Performance Solutions.* Addison-Wesley, 2001.
20. D. Sosnoski. *XML and JAVA technologies: Document models, Part 1: Performance.* `http://www-128.ibm.com/developerworks/xml/library/x-injava/`.
21. D. Sosnoski. *XMLBench Document Model Benchmark.* `http://www.sosnoski.com/opensrc/xmlbench/`.
22. The Crimson Java Parser. `http://xml.apache.org/crimson/`.
23. The TimeNET tool. `http://pdv.cs.tu-berlin.de/~timenet/`.
24. The Xerces Java Parser. `http://xerces.apache.org/xerces-j/`.
25. The Xerces2 Java Parser. `http://xerces.apache.org/xerces2-j/`.
26. V. Datla and K. Goševa-Popstojanova. Measurement-based Performance Analysis of E-commerce Applications with Web Services Components. In *IEEE ICEBE*, pages 305–314, 2005.
27. T. Verdickt, B. Dhoedt, F. Gielen, and P. Demeester. Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. *IEEE Trans. Softw. Eng.*, 31(8):695–711, 2005.

28. C. Woodside and D. Menascé. Application-Level QoS. *IEEE Internet Computing*, 10(3):13–15, 2006.
29. C. Woodside, J. Neilson, D. Petriu, and S. Majumdar. The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. *IEEE Trans. Computers*, 44(1):20–34, 1995.
30. C. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (PUMA). In *ACM WOSP*, pages 1–12, 2005.