Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 261 (2010) 147–162
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

# A Novel Cost Model of XML Serialization

G. Imre[a,1], M. Kaszó[a,2], T. Levendovszky[b,3] and H. Charaf[a,4]

[a] *Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Budapest, Hungary*

[b] *Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN USA*

**Abstract**

Using XML as a data representation format is a common choice when integrating software systems on different platforms. The serialization of in-memory object instances of a class into corresponding XML documents heavily influences the performance of the XML-based communication, even if we send the XML over HTTP as in the case of SOAP-based XML Web Services, or with asynchronous messaging such as Java Message Service (JMS), or simply saving it into a file. Several studies have been published analyzing the performance impact of XML serialization on different platforms. No models or measurement methodologies have been proposed however, to establish a relationship between the serialization cost of primitive types (e.g. int, double, string), and the serialization cost of composite types. Such a model can be very useful when the type of the XML messages exchanged during the communication are known a priori, recorded in an interface definition, similarly to the Web Services Description Language (WSDL) in case of XML Web Services. This paper introduces a model that is validated with measurements on .NET and Java platform. The opposite direction, deserialization is covered as well. The main mathematical tool used is linear regression, but cases are also shown and explained where linearity is compromised.

*Keywords:* Software performance, XML serialization, performance modeling

## 1 Introduction

Service-oriented architecture (SOA) for enterprise systems is a relatively new, but rapidly proliferating paradigm in software engineering. The business functionality of the individual systems is published as services with well-defined interfaces that are specified in a standard, platform-neutral way, such as Web Services Description Language (WSDL). Using elementary services, composite services can be created, or even complex workflows can be orchestrated that are capable of partly or completely automating business processes. Among the several advantages of SOA based systems, we find flexibility, platform-independence and better reuse. However the

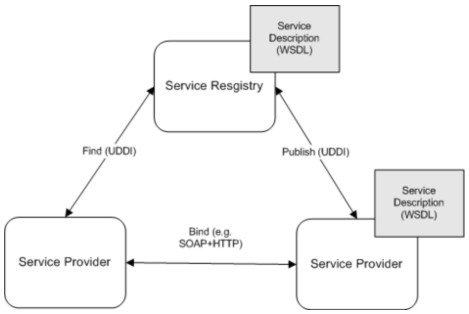
1 Email: gabor@aut.bme.hu
2 Email: mkaszo@aut.bme.hu
3 Email: tihamer@isis.vanderbilt.edu
4 Email: hassan@aut.bme.hu

doi:10.1016/j.entcs.2010.01.010

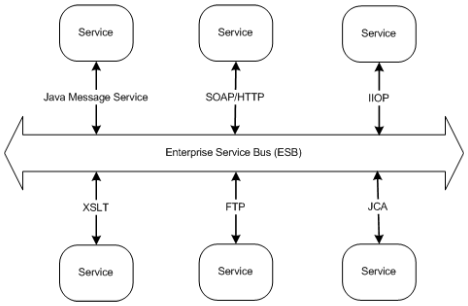
performance prediction of such systems is still a challenge, since a great number of systems can be involved while serving a request. The architects, developers or maintainers of a SOA-based system often face the following questions: How will the response time of the system change if the processor in one of the servers is upgraded to double speed? What kind of performance gain can be achieved by increasing the network bandwidth by a factor of 1.5? To what extent will the performance degrade if the number of concurrent clients grows by 30 percent? Which data representation format and communication protocol should the systems use?

This latter question is reasonable if we consider that the SOA principles do not mandate the usage of classic XML Web Services, which exchange data in XML format, wrapped in a SOAP (Simple Object Access Protocol) envelope, over HTTP. The only requirement is that the Service Provider publishes a description of the service in WSDL, as explained in Figure 1. It can be published in a Service Registry, most often supporting the Universal Description Discovery and Integration (UDDI) standard. Service Requestors either find the WSDL file in the Service Registry, or the Service Provider shares it in an alternative way. The WSDL description may include different type of *bindings*, which describe the protocols through which the client can access the service. It is possible that a background system is only accessible via a binary communication protocol such as IIOP (Internet Inter-Orb Protocol) or through asynchronous messaging, and the data is represented in simple comma-separated value (CSV) format. This variety of protocols and data formats is often abstracted away using the Enterprise Service Bus (ESB) architectural pattern depicted in Figure 1. Both open source and commercial implementations of this pattern support several protocols, which allows the developers to simply put a message on the bus, configure the destination, and all necessary transformation or protocol conversion is performed by the ESB. Additional aspects such as security, logging, and monitoring are also supported by the ESBs.

We can use three main methods to answer questions similar to the ones above: measurement, simulation or analytical performance modeling. Measurements provide the most accurate results, however they can be carried out only after developing the system to be tested. Furthermore, the execution of tests can take considerable amount of time, if we want to consider more factors that can influence the perfor-



(a)



(b)

Fig. 1. (a) The Participants of a Web Service Call (b) The Enterprise Service Bus Architectural Pattern

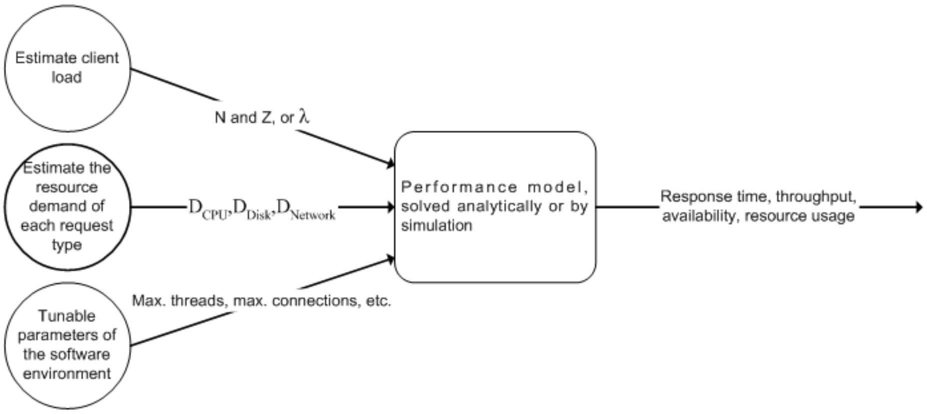

Estimate client load
N and Z, or λ
Estimate the resource demand of each request type
$D_{CPU}, D_{Disk}, D_{Network}$
Tunable parameters of the software environment
Max. threads, max. connections, etc.
Performance model, solved analytically or by simulation
Response time, throughput, availability, resource usage

Fig. 2. The usual input and output of performance models

mance. The main advantage of analytical performance models is that the definition, the parametrization, and the evaluation of a performance model have relatively low cost. In contrast to several hours or even days of an exhaustive load testing, a performance model can provide the necessary results in seconds. However, the accuracy of analytical models is lower than that of measurements, since models always disregard some portions of the modeled system. A common technique for establishing performance models use queueing networks as described by several authors [9,13,15]. Solving such models can predict the relevant performance metrics (e.g. response time, throughput) using some input parameters of the system under investigation, as illustrated by Figure 2. Some of the input parameters characterize the load offered by the clients. In closed systems, the number and the average think time of the clients ($N$ and $Z$), while in open systems the rate of arrivals ($\lambda$) can serve this purpose. Some performance models [17,12] can take some tunable parameters of the software environment into account, such as the maximum number of threads or network connections. The third group of input parameters characterizes the resource demand (e.g. processor, disk, network) i.e. the time needed at each of the resources to serve the different types of requests. The queueing network of a system can be created in the design phase of the software life cycle, but the exact resource demand can only be measured in the implemented system. Being able to solve the performance model *before* the implementation, however, could produce valuable results that influence the implementation decisions. For this reason, performance experts often estimate the resource demand of serving client requests. We propose a method, by which this estimation can be quite precise, regarding the communication overhead of the request, in case of a priori knowledge about the message type and size.

We investigate the serialization overhead of XML-based communication. As we mentioned, SOA does not require the XML data format, but it is still a very common choice due to its platform-independence and its ability to represent complex data structures. To represent data in XML, the in-memory objects need to be converted into an XML document (or fragment), this process is called XML serialization that requires CPU cycles, before sending the data over the network. The cost of XML serialization is hence composed of the CPU and network demand. In service-oriented applications, the service interfaces, including the message types, are described in WSDL, before the services are implemented. We show that using only the WSDL, and an estimation about the length of the dynamic data structures such as collections, the cost of XML serialization on a concrete system can be estimated if we measure this cost only for primitive types on that concrete system.

The rest of this paper is organized as follows. Section 2 summarizes related work. Section 3 contains the explanation and the validation of the cost model of XML serialization. Finally, conclusions are drawn in Section 4, and future work is presented.

## 2 Related Work

In the past years several techniques and methods were proposed to address performance prediction using performance modeling. A group of them are based on queueing networks [11], or extended or layered versions of queueing networks [16]. These methods establish a queueing network model of the system. By solving this model with analytical methods or simulation, the prediction of performance metrics is possible. Some of the proposed methods generate a queueing network model of the system based on its UML model [7,8]. Urgaonkar et al. [17] present a queueing model for multi-tier internet applications, where queues represent different tiers of the application. The model faithfully captures several aspects of web applications, such as caching and concurrency limits at the tiers. Another group of performance modeling techniques uses Petri nets or generalized stochastic Petri nets, such as [5]. Petri nets can represent blocking and synchronization aspects much more than queueing networks, which are more suitable for modeling resource contention and scheduling strategies. A powerful combination of the queueing network and the Petri net formalism is presented by Kounev and Buchmann [12]. Using queueing Petri nets, the authors successfully model the performance of a web application, considering the maximum size of thread pools. Performance modeling of service-oriented systems is a more recent topic with less widely-spread solutions than in the case of traditional web applications. Ardagna et al. propose a model-driven approach, transforming a design model of service composition into an analysis model, used by a probabilistic model checker for quality (service execution time, service cost and reliability) prediction [4]. These quality dimensions are discussed in work by others as well [6,18]. Menascé and Dubey design, implement and evaluate a Quality of Service (QoS) broker that selects appropriate services based on the service requestor's preference regarding response time, throughput and availability [14]. All of the referenced approaches could benefit of a cost model of the communication between the service requestor and provider that is capable of calculating the resource demand. Our work aims to establish such a cost model, firstly for XML based communication.

The performance of XML serialization has been analyzed in several papers, however not with our goal in view, i.e. they do not provide a cost model that can feed a queueing network-based performance model. Juric et al. [10] compare the functionality and performance of Web services to Remote Method Invocation (RMI), which is a binary communication protocol for Java. It also discusses the secure variants, WS-Security and RMI-SSL, both on Windows and Linux platforms. They conclude that RMI is an order of magnitude faster than Web services. This result in itself does not mean however that an XML-based service is always about 10 times slower than a service accessible through a binary protocol. If the processing of a service request takes a considerable amount of time (e.g. complex database queries), the communication part of the whole response time, and so the difference resulted from using the two protocols becomes negligible. The complete performance model that we propose helps deciding if that is the case, or the communication is a bottleneck and the usage of binary protocols should be considered, sacrificing platform-independence. In such cases, not only RMI is an option, but there are other binary protocols optimizing web services performance, such as Hessian [2] or Fast Infoset [1].

## 3 Contributions

The following questions are relevant when we want to take the overhead of XML serialization in a performance model of a service-oriented application into account: How does the cost of serialization depend on the data type of the messages to be serialized? How does the serialization overhead depend on the length of a dynamic data structure, such as an array? How does the length of a string affect the cost of its serialization? How can we estimate the serialization overhead of a known composite data type if we know the serialization cost of the composing primitive data types? To answer these questions, we performed measurements, built a model, and validated the goodness of fit against the measurement results, as presented in subsections 3.1, 3.2 and 3.3 respectively.

The test cases that we measured are implemented on two platforms: Microsoft .NET 3.5, using the C# programming language and Java Standard Edition 6. We investigate these platforms, since they are modern and popular, and they are often chosen when implementing service-oriented systems.

The first part of the test cases serializes a specific object $10,000$ times into the same network stream (i.e., sends the serialized XML over the network), and measures the average execution time of the serialization. The measurement is performed with the built-in *System.Diagnostics.Stopwatch* class in .NET, and with the *hrtlib* library in Java that provides a high resolution timer using the native timer capabilities of the operating system. The second part of the test cases measures the cost of deserialization by serializing an object into the memory, and deserializing it back $10,000$ times. The CPU usage during the test cases is recorded as well, the CPU and I/O cost can be separated this way. The test cases were executed on a PC with Microsoft Windows Server 2003 R2 Service Pack2, a 2.4 GHz Intel Pentium 4 processor and 1 GB memory. The target of the serialization is another PC with Windows XP, and a 3 GHz Intel Pentium 4 HyperThreading processor and 2 GB memory. The machines are connected via a 100 Mbit/s switched LAN. Both the

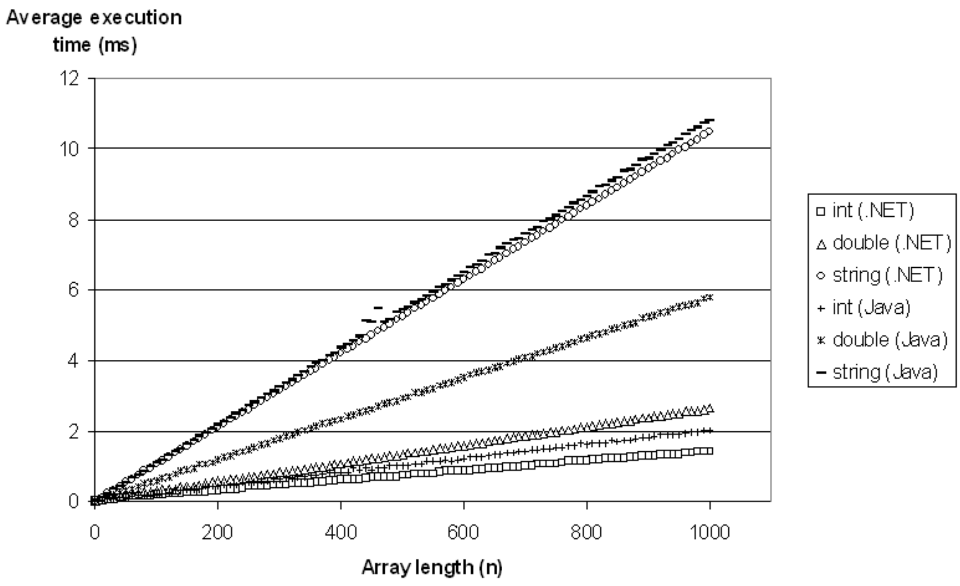


Fig. 3. The Cost of XML Serialization as the Function of Array Length

Java Runtime Environment (Sun JDK 6 Update 14) and the .NET Common Language Runtime were run in their default configurations. The following subsections detail each of the measurements and the models deducted from them.

## *3.1 Effect of the Array Length*

We measured six test cases to investigate the dependence of the serialization cost on the array length on both platforms. We serialize and deserialize an array of integers, an array of doubles and an array of strings. The length of the array ($n$) is changed between 1 and 1000, in a 10 step. The length of the string is fixed at 100, because the effect of the string length is examined separately. The results of the measurements are depicted in Figure 3 and Figure 4. It can be clearly seen that both the serialization and deserialization cost is a linear function of the array length, with array of integers having the lowest and array of strings having the highest gradient. The .NET platform performs better in all of the test cases. Due to the linearity, the resource demand $D$ can be expressed as the function of the type ($t$) and the length ($n$) of the array.

(1) $$D = c_0^t + c_1^t * n, \quad t \in \{int, double, string\} \times \{.NET, Java\} \times \{ser, deser\}$$

To find the type-dependent $c_0^t$ and $c_1^t$ coefficients, we applied linear regression as the best linear estimator on the data. Table 1 summarizes the coefficients provided by the linear regression, and it contains the coefficient of determination ($R^2$) as well. It can be seen that this value is very close to 1, indicating an excellent fit of the linear model. The linearity itself is not a surprising result, but determining these coefficients is important when we try to find a general formula to calculate the resource demand of composite types in subsection 3.3.

Table 1
The Result of the Linear Regression, Using the Array Length as the Regressor

| $t$ | $c_0^t$ | $c_1^t$ | $R^2$ |
|---|---|---|---|
| (int, .NET, ser) | 0.0537 | 0.0014 | 0.9993 |
| (double, .NET, ser) | 0.0455 | 0.0023 | 0.9999 |
| (string, .NET, ser) | 0.0231 | 0.0105 | 1.000 |
| (int, Java, ser) | 0.0464 | 0.0020 | 0.9993 |
| (double, Java, ser) | 0.0521 | 0.0058 | 0.9999 |
| (string, Java, ser) | 0.0292 | 0.0108 | 0.9996 |
| (int, .NET, deser) | 0.0402 | 0.0015 | 0.9998 |
| (double, .NET, deser) | 0.046 | 0.0021 | 0.9999 |
| (string, .NET, deser) | 0.0583 | 0.0035 | 0.9998 |
| (int, Java, deser) | 0.0912 | 0.0023 | 0.9990 |
| (double, Java, deser) | 0.1062 | 0.0033 | 0.9988 |
| (string, Java, deser) | 0.0212 | 0.0043 | 0.9974 |

## *3.2 Effect of the String Length*

In this section, we investigate how the length of a string ($l$) affects its serialization cost. Our initial test case was similar to the case of the array length: a string, whose length is varied from 1 to 1000 in 10 steps, is serialized and deserialized. Figure 5 points out that the curves for serialization contain a cut-off point. This point is more obvious for .NET around string length 300, but it can be observed in Java as well, around string length 120. This suggests that this relationship could be better expressed with two lines than with one. Questions arise however: What is the cause of this cut-off point and where it is exactly? Is it possible that the curve contains further cut-off points for string length values higher than 1000? To answer these questions, we investigated the source code of the .NET Framework and the Java Runtime Environment. The Sun JDK 6 is open source, the relevant parts of the .NET Framework can be disassembled (decompiled) using .NET Reflector [3]. We found that the writing of characters to the output stream is buffered at multiple points. In .NET for example, a buffer of 256 and a buffer of 1024 bytes are used. To prove that this causes the more flat line for small string lengths, we wrote a similar code to that found in the .NET Framework. It turned out that removing the 256 sized buffer removes the cut-off point, hence we conclude that the buffering causes the better performance of the XML serialization for small string lengths, and we place the cut-off point at 256. In Java, a similar buffer is used with a size of 128. The lines for deserialization are less steep, and do not contain any cut-off points in

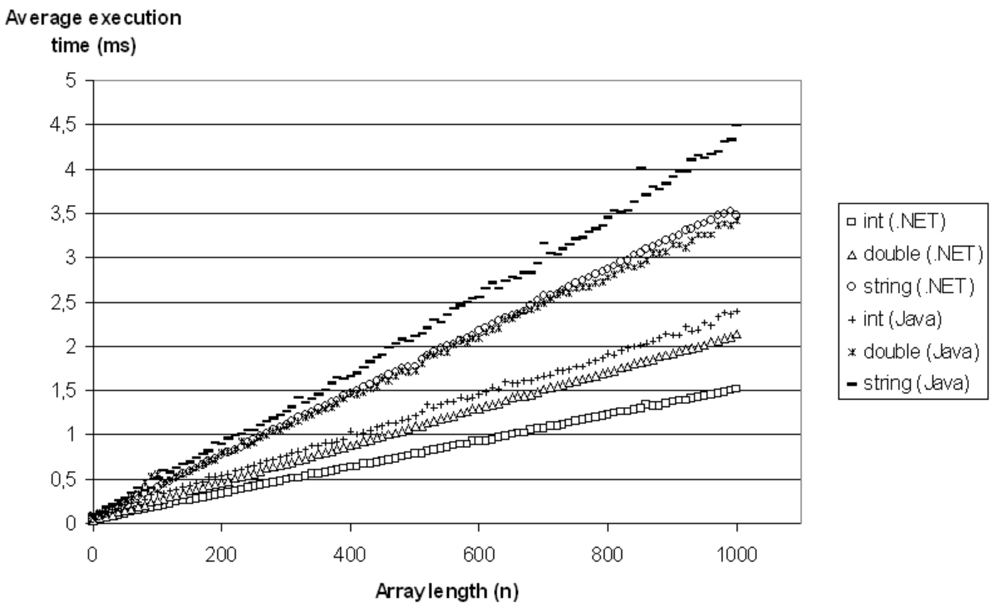


Fig. 4. The Cost of XML Deserialization as the Function of Array Length

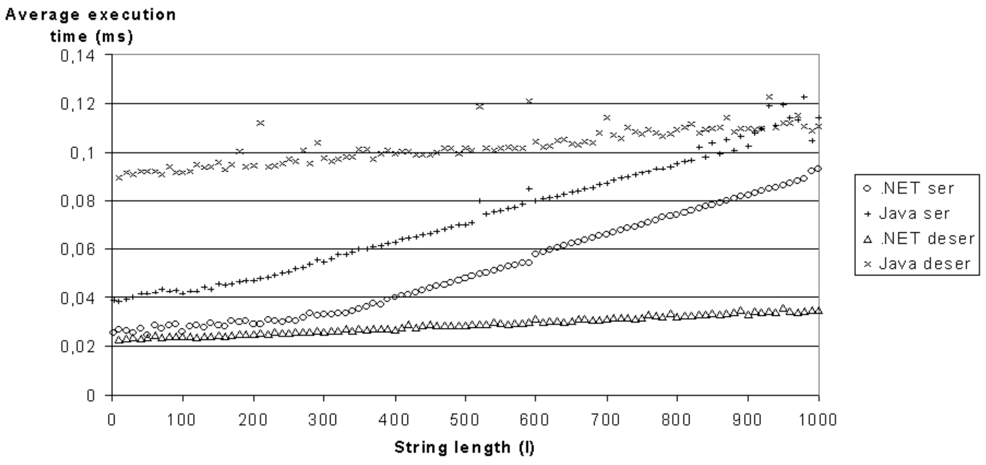


Fig. 5. The Cost of XML Serialization and Deserialization as the Function of String Length

this interval.

To make sure there are no further cut-off points, we repeated our measurements until a string length of 10000. As Figure 6 illustrates, the gradient of the curve remains constant for the .NET serialization case. The other curves, however, change their behavior for higher string lengths. In the case of deserialization, the curve is only piecewise linear, due to the buffer that is used to store the characters of the string read between the start and end tags in the XML. When this buffer is full, the allocation of new memory area and the copying of the existing data has an overhead

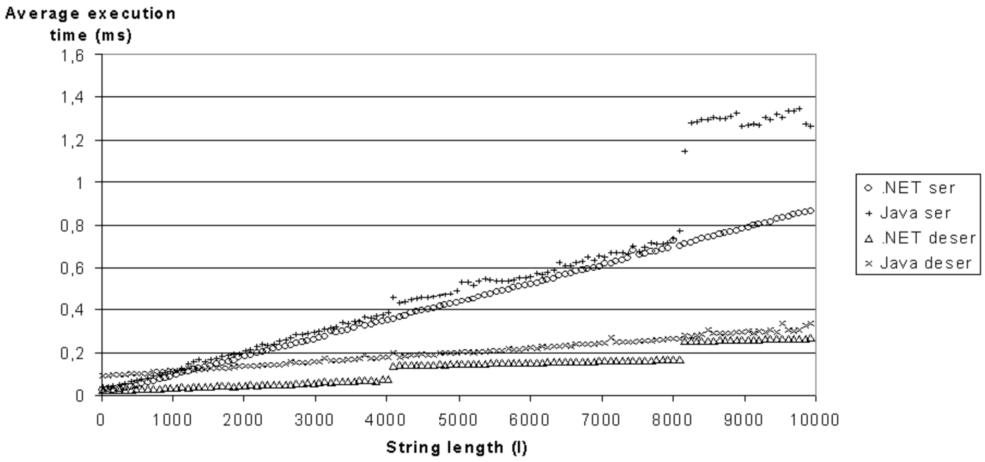


Fig. 6. The Cost of XML Serialization and Deserialization as the Function of String Length

causing the jump in the deserialization curves. In the .NET case, this buffer size is 4096 and can be clearly identified in Figure 6. In Java, the buffer size is 8192, and can be hardly recognized because it is a relative low overhead compared to the total cost of deserialization. Surprisingly, the output buffering (with 8192 buffer size as well) in the case of Java serialization has a more significant overhead.

Table 2
The Result of the Linear Regression, Using the String Length as the Regressor

| $t$ | $l$ | $c_0^{t,l}$ | $c_1^{t,l}$ | $R^2$ |
|---|---|---|---|---|
| (.NET, ser) | $l \leq 256$ | 0.0261 | 1.36e-5 | 0.6344 |
| (.NET, ser) | $256 < l \leq 10000$ | 0.0060 | 8.68e-5 | 0.9993 |
| (Java, ser) | $l \leq 128$ | 0.039 | 3.72e-5 | 0.6883 |
| (Java, ser) | $128 < l \leq 8160$ | 0.0336 | 8.91e-5 | 0.9954 |
| (Java, ser) | $8160 < l \leq 10000$ | 1.069 | 2.4e-5 | 0.1240 |
| (.NET, deser) | $l \leq 4060$ | 0.0228 | 1.25e-5 | 0.9902 |
| (.NET, deser) | $4060 < l \leq 8160$ | 0.1109 | 7.04e-6 | 0.9945 |
| (.NET, deser) | $8160 < l \leq 10000$ | 0.1969 | 6.96e-6 | 0.9559 |
| (Java, deser) | $l \leq 8160$ | 0.0916 | 2.16e-5 | 0.9974 |
| (Java, deser) | $8160 < l \leq 10000$ | 0.1314 | 1.80e-5 | 0.8222 |

Due to the buffering effects explained above, separate linear regressions should be applied for the different intervals determined by the buffer sizes. This implies that the resource demand ($D$) of serializing a string to XML is estimated as a function of the string length ($l$) this way:

$$D = c_0^{t,l} + c_1^{t,l} * l, \quad t \in \{.NET, Java\} \times \{ser, deser\} \tag{2}$$

This means that the coefficients have different values depending on the length of the string, as summarized in Table 2. The linear fit is in most of the cases excellent, with high $R^2$ value. Outlier removal was applied only at Java deserialization. Lower $R^2$ can be observed for serialization with small string lengths, where the regression line declines more from the measured values.

### *3.3 Serialization and Deserialization Cost of Composite Types*

A composite type is a data type that encapsulates several fields that are primitive or other composite data types. In programming languages, a composite type can be represented as a struct, class or similar language construct. In XML, the fact that a composite type owns a field is naturally expressed by adding the field as a child element or an attribute. To investigate the serialization cost of composite data types, we defined C# and Java classes that contain 0 to 10 integer, double and string fields respectively, which means $11^3 - 1 = 1330$ types in each language. (The case where no fields are included at all is disregarded.)To generate the source code of these files, a small script was developed. Each test case serialized an instance of each composite type. All the fields of type string held 100, 500 or 1000 long strings, which amounts to $3 * 1330 = 3990$ test cases. (To decrease the number of necessary test cases, we made all the string fields equal length, instead of measuring for all possible string length combinations for the different fields.) This amount of measurement results is hard to represent in a graphical way, since each measured result depends on four variables: the number of integer fields ($i$) in the composite type, the number of double fields ($d$), the number of string fields($s$) and the length of the string fields ($l$). The fact that the serialization cost showed a linear behavior as a function of the array length suggests that the same can hold for the serialization cost of composite types. Arrays can be considered after all as composite objects, with fields of homogenous type. We also know that the linear assumption does not hold for the dependence on the string length because of buffering effects. For this reason, we perform a multiple linear regression, that is, we try to express the resource demand ($D$) of the serialization and deserialization as

$$D = c_0^{t,l} + c_{int}^{t,l} * i + c_{double}^{t,l} * d + c_{string}^{t,l} * s, \quad t \in \{.NET, Java\} \times \{ser, deser\} \tag{3}$$

We introduced the $c$ coefficients as dependent on the string length $l$, since we want to perform separate linear regressions for the different string sizes. Table 3 contains the results of the multiple linear regression. We also indicated the 95% confidence intervals for the coefficients that are quite narrow in most of the cases (mostly under 5% with 13% being the highest) which suggests statistically significant results. The $R^2$ values are also very close to 1, indicating an excellent fit of the linear model. The table contains the $F$ values (ratio of explained and unexplained variance) for all three cases. They must be compared to the critical value (2.611613) of the $F$ distribution with model degrees of freedom 3 (the number of regressor variables) and error degrees of freedom $1330 - 3 - 1 = 1326$, because the number of samples

Table 3
The Result of the Multiple Linear Regression for Composite Types

| $t$ | $l$ | $c_0^l$ | $c_{int}^l$ | $c_{double}^l$ | $c_{string}^l$ | $R^2$ | $F$ |
|---|---|---|---|---|---|---|---|
| (.NET, ser) | 100 | 4.73e-2 | 1.41e-3 | 2.36e-3 | 6.08e-3 | 0.9445 | 7.528 |
| (.NET, ser) | 500 | 3.2e-2 | 1.26e-3 | 2.02e-3 | 4.25e-2 | 0.9950 | 8.741 |
| (.NET, ser) | 1000 | 2.93e-2 | 1.41e-3 | 2.25e-3 | 8.62e-2 | 0.9988 | 3.576 |
| (Java, ser) | 100 | 5.45e-2 | 2.81e-3 | 5.11e-3 | 8.23e-3 | 0.9627 | 3.249 |
| (Java, ser) | 500 | 5.34e-2 | 2.70e-3 | 3.71e-3 | 4.38e-2 | 0.9956 | 9.874 |
| (Java, ser) | 1000 | 5.46e-2 | 3.07e-3 | 3.13e-3 | 8.74e-2 | 0.9983 | 2.924 |
| (.NET, deser) | 100 | 3.86e-2 | 1.82e-3 | 2.41e-3 | 2.90e-3 | 0.9938 | 7.062 |
| (.NET, deser) | 500 | 3.71e-2 | 1.85e-3 | 2.45e-3 | 8.71e-3 | 0.9907 | 4.688 |
| (.NET, deser) | 1000 | 4.40e-2 | 1.68e-3 | 2.02e-3 | 1.48e-2 | 0.9846 | 2.817 |
| (Java, deser) | 100 | 1.19e-1 | 2.77e-3 | 4.05e-3 | 4.34e-3 | 0.8892 | 3.422 |
| (Java, deser) | 500 | 1.18e-1 | 2.99e-3 | 4.08e-3 | 1.16e-2 | 0.9466 | 7.815 |
| (Java, deser) | 1000 | 1.21e-1 | 2.90e-3 | 4.17e-3 | 2.11e-2 | 0.9854 | 2.653 |

is 1330. In all cases, the $F$ values are greater than the critical value at significance level 5%, meaning that our linear model is adequate.

To test the fit of the data, graphical methods are used as well. The scatter plot of the residuals (the difference between the measured values and the values predicted by the model) versus the predictor variables $(i, d, s)$ can be seen in Figure 7 and 8 in case of the string fields of length 100 for .NET serialization. For the number of integer and double fields, no systematic pattern can be noticed, and the scatter in the residuals are similar for all levels of the predictor variables which confirms the linear model. For the number of string fields, a slight pattern can be observed, but the variation is constant across the data here as well. Similar scatter plots were constructed for the other cases as well. In the case of Java serialization and deserialization, the scatter plots showed some outliers (always less then 5% of the measured points), such as the four points in Figure 8. The possible cause for this is a long running garbage collection. Table 3 contains the results after the removal of the outliers.

At this point it is worth comparing the coefficients to the ones resulted in the previous subsections (Tables 1 and 2).

In Figure 9 and 10, one can observe that with respect to the confidence intervals, the coefficient $c_{int}^l$ is equal for all the investigated string lengths, and equals the $c_1^{int}$ value in Table 1 in most cases. The same holds for $c_{double}^l$ which can be considered independent from $l$ and equal to $c_1^{double}$ in Table 1. The only exception is Java

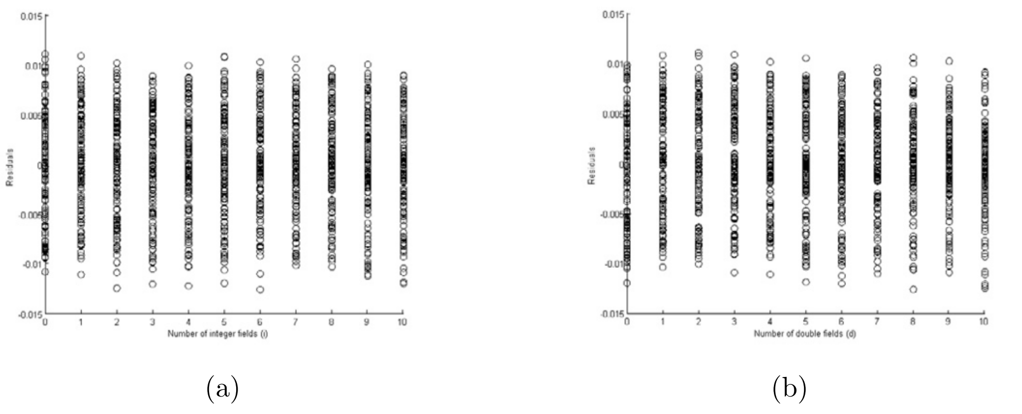


Fig. 7. Scatter Plot of the Residuals Versus the Number of Fields for .NET Serialization (a) Integer (b) Double

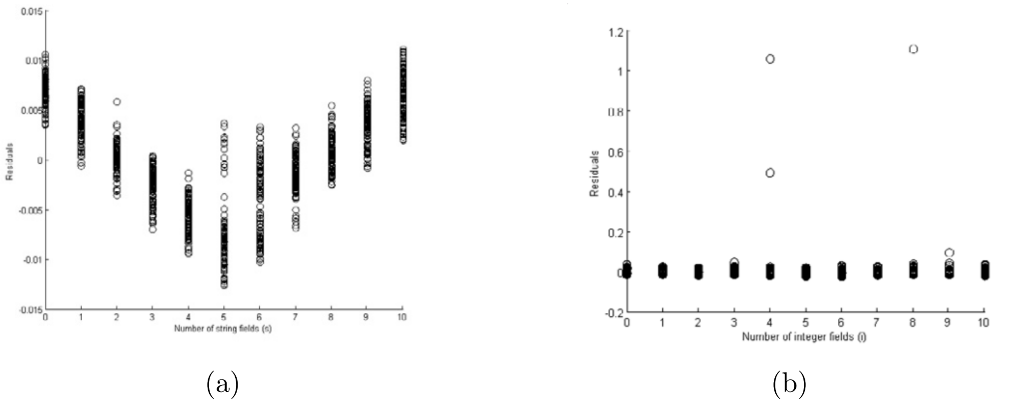


Fig. 8. Scatter Plot of the Residuals (a) Versus the Number of Fields for .NET Serialization (b) Versus the Number of Integer Fields for Java Deserialization, Showing Outliers

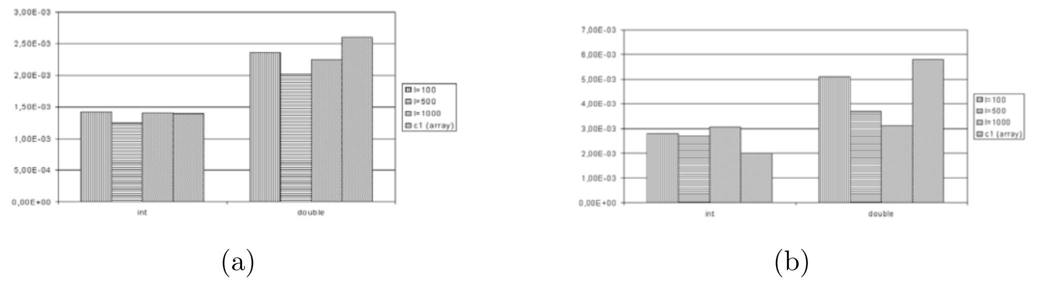
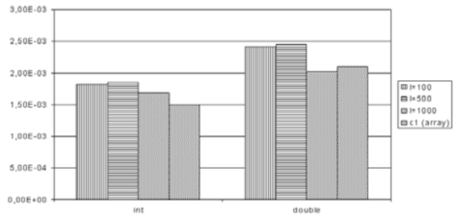
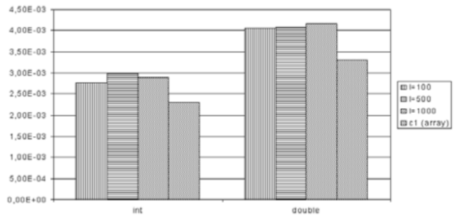


Fig. 9. The Coefficients for the Serialization of Arrays and Composite Types (a) .NET (b) Java

serialization, where both $c_1^{(int,Java,ser)}$ and $c_1^{(double,Java,ser)}$ differ significantly from the coefficients retrieved with the multiple linear regression for composite types. For the other three cases (.NET serialization and deserialization, and Java deserialization) our starting point is confirmed, namely, that the serialization cost of one additional integer or double value (represented by the appropriate coefficients) is the same, the value is either a member in an array or a field in a composite object.



(a)



(b)

Fig. 10. The Coefficients for the Deserialization of Arrays and Composite Types (a) .NET (b) Java

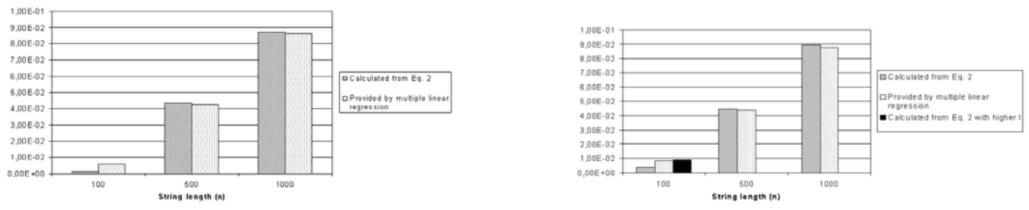
This observation has a very important practical implication as well. If we want to know the cost of XML serialization or deserialization on a given machine for any composite type containing integer and double fields, it is enough to measure the serialization or deserialization cost of the array for some lengths (in our case $1, 10, 20, ...1000$), perform a linear regression, and the resulting coefficients can be reused in calculating the serialization overhead of any composite object.

Naturally, in the case of strings, the coefficients in Table 3 are not independent from the string length. But Equation 2 can provide this length-dependent coefficient. We calculate the additional serialization cost of one string of length $l$ as $c_1^{t,l} * l$ with the $c_1^{t,l}$ value in Table 2. Figure 11 and 12 compare these calculated values to the $c_{string}^{l}$ values given by the multiple linear regression. We should note that at serialization, for small string length (100) the two values differ to an unacceptable measure, which coincides with the fact that the linear fit is worse for small string lengths, with a smaller $R^2$ value in Table 2. For this reason, we also indicated the coefficient calculated with $c_1^{t,l}$ with higher $l$ values. As Figure 11 shows, the result is closer to that given by the multiple linear regression. This suggests that when serializing composite types, the buffering effects for small string lengths do not appear.

Finally, we take a closer look at the resulting constant values ($c_0$) of the different regressions. This constant represents a fix cost of serialization or deserialization, since some operations need to be performed, independently from the number of objects to be serialized. We expect that summing the $c_0$ constants of the first regression (the array length being the regressor) for the different types (see Table 1), the result will be higher than the fix cost for composite types (Table 3). The cause of this is that some of the helper objects are reused during the serialization of the different fields of the composite object. Figure 13 and 14 confirm this expectation. We can observe that the $c_0^{int}$ obtained from the array test cases is quite close to the $c_0$ values resulted for composite types.

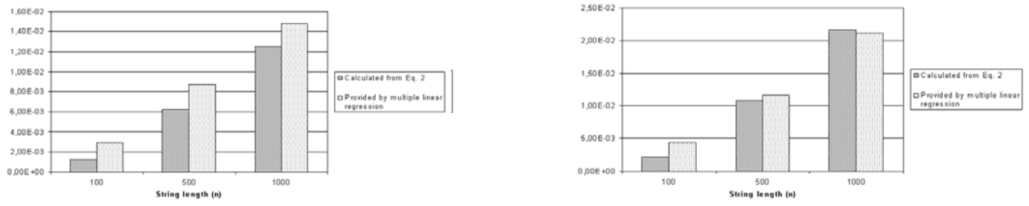
## *3.4 The Proposed Method*

Equations 1, 2 and 3 summarize the proposed linear models for the cost of XML serialization and deserialization. In order to apply it, the architect or developer designs the input and output message types of a service interface. From these type definitions, the number of integer, double, and string fields ($i, d, s$ respectively) can



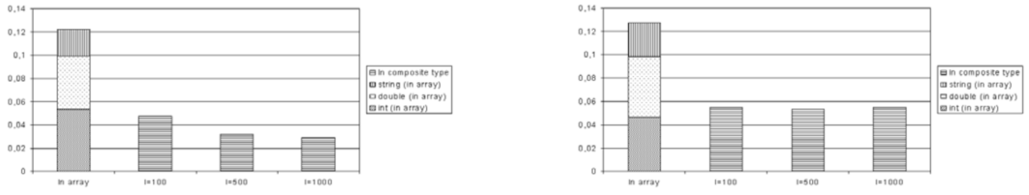
(a) (b)

Fig. 11. The Coefficients of Strings for the Serialization of Strings and Composite Types (a) .NET (b) Java



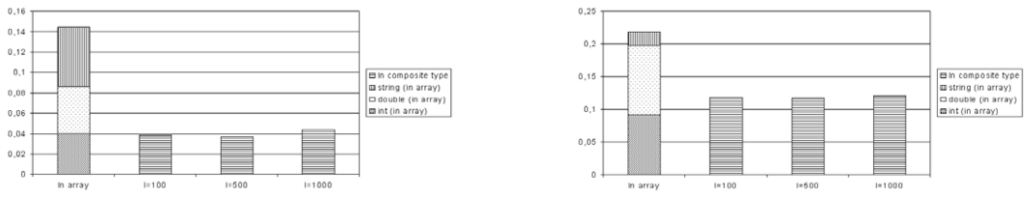
(a) (b)

Fig. 12. The Coefficients of Strings for the Deserialization of Strings and Composite Types (a) .NET (b) Java



(a) (b)

Fig. 13. Comparing the $c_0$ Coefficients for the Serialization of Arrays and Composite Types (a) .NET (b) Java



(a) (b)

Fig. 14. Comparing the $c_0$ Coefficients for the Deserialization of Arrays and Composite Types (a) .NET (b) Java

be derived. Then, the length of dynamic data structures ($n$) and the length of strings ($l$) has to be estimated. Next, the coefficients appearing in the equations have to be identified on a given machine. As a first step, the test cases for arrays of integer and double have to be measured, and linear regression has to be performed on the results. This gives the $c_i^t$ coefficients where $i \in \{0, 1\}, t \in \{int, double, string\} \times$

$\{.NET, Java\} \times \{ser, deser\}$. Next, we execute the test cases for string length and identify the cut-off points and jumps caused by the buffering. Performing separate linear regressions on the linear sections, we obtain $c_i^{t,l}$, $i \in \{0,1\}, t \in \{.NET, Java\} \times \{ser, deser\}$. Moving to Equation 3, we have shown that

$$c_{int}^{t,l} = c_1^{int,t}, \tag{4}$$

and

$$c_{double}^{t,l} = c_1^{double,t}, \tag{5}$$

where the right sizes are already identified in the first test case, independent from $l$. $c_{string}^{t,l}$ can be derived from the string length test case, namely

$$c_{string}^{t,l} = c_1^{t,l'} * l \tag{6}$$

and

$$c_{string}^{t,l} = c_1^{t,l'} * l. \tag{7}$$

In these equations the notation $l'$ refers to the fact that in case of small string lengths, not the original $l$ should be considered when choosing the coefficients from Table 2, but a larger $l'$, for which the buffering effect of small strings does not appear. Finally, we have to determine the $c_0^{t,l}$ coefficient in Equation 3. As we have seen, the sum $c_0^{int,t} + c_0^{double,t} + c_0^{string,t,l}$ overestimates the value provided by the multiple linear regression, but choosing

$$c_0^{t,l} = c_0^{int,t} \tag{8}$$

is an acceptable estimation.

## 4 Conclusions and Future Work

This paper presented a series of performance measurements that investigated the cost of XML serialization and deserialization on .NET and Java platform. In all cases the .NET platfrom proved to be faster, but we have to emphasize that our goal was not the fair comparision of the two platforms. We did not pay attention to optimally tune the runtime environments, both of them worked in their default configurations. Linear regression was used to analyze the results, and we found that the linear model is appropriate for predicting the serialization cost as a function of number of primitive types in the serialized and deserialized array or composite object on both platforms. If the primitive type is a string, and we consider the cost as the function of the string length, the output or input buffering has a non-linear effect on the serialization or deserialization cost. We proposed a method how these results can be used to identify the relevant coefficients on a given machine with a limited set of measurements. Using these coefficients, the sizes of dynamic structures, and the string lengths, the serialization and deserialization overhead of any type of XML message can be estimated. The estimation is quite precise for most of the cases. Our future goal is to apply this cost model in a complex, queueing network based performance model. The cost of XML serialization and deserialization estimated with our method can serve as an input parameter in such models, achieving more precise early performance prediction of SOA-based systems.

## References

[1] *Home page of the fast infoset project*, https://fi.dev.java.net/.

[2] *Home page of the hessian binary web service protocol*, http://hessian.caucho.com/.

[3] *Home page of the .NET reflector*, http://www.red-gate.com/products/reflector/.

[4] Ardagna, D., C. Ghezzi and R. Mirandola, *Model driven QoS analyses of composed web services*, in: *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet* (2008), pp. 299–311.

[5] Bernardi, S., S. Donatelli and J. Merseguer, *From uml sequence diagrams and statecharts to analysable petri net models*, in: *WOSP '02: Proceedings of the 3rd international workshop on Software and performance* (2002), pp. 35–45.

[6] Chandrasekaran, S., J. A. Miller, G. A. Silver, I. B. Arpinar and A. P. Sheth, *Performance analysis and simulation of composite web services*, Electronic Market:The Intl. Journal of Electronic Commerce and Business Media **13** (2003).

[7] Cortellessa, V., A. D'Ambrogio and G. Iazeolla, *Automatic derivation of software performance models from case documents*, Performance Evaluation **45** (2001), pp. 81–105.

[8] Cortellessa, V. and R. Mirandola, *Deriving a queueing network based performance model from uml diagrams*, in: *WOSP '00: Proceedings of the 2nd international workshop on Software and performance* (2000), pp. 58–70.

[9] Jain, R., "The Art of Computer Performance Analysis," John Wiley & Sons, 1991.

[10] Juric, M. B., I. Rozman, B. Brumen, M. Colnaric and M. Hericko, *Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL*, Journal of Systems and Software **79** (2006), pp. 689–700.

[11] Kleinrock, L., "Theory, Volume 1, Queueing Systems," Wiley-Interscience, 1975.

[12] Kounev, S. and A. Buchmann, *Performance modelling of distributed E-Business applications using queuing petri nets*, in: *Proc. of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'03)*, 2003.

[13] Lazowska, E. D., J. Zahorjan, G. S. Graham and K. C. Sevcik, "Quantitative system performance: computer system analysis using queueing network models," Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984, 152-176 pp.

[14] Menasce, D. A. and V. Dubey, *Utility-based QoS brokering in service oriented architectures*, in: *Web Services, 2007. ICWS 2007. IEEE International Conference on*, 2007, pp. 422–430.
URL http://dx.doi.org/10.1109/ICWS.2007.186

[15] Menascé, D. A. and V. Almeida, "Capacity Planning for Web Services: metrics, models, and methods," Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[16] Rolia, J. A. and K. C. Sevcik, *The method of layers*, IEEE Trans. Softw. Eng. **21** (1995), pp. 689–700.

[17] Urgaonkar, B., G. Pacifici, P. Shenoy, M. Spreitzer and A. Tantawi, *An analytical model for multi-tier internet services and its applications*, SIGMETRICS Perform. Eval. Rev. **33** (2005), pp. 291–302.

[18] Zeng, L., B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, *QoS-aware middleware for web services composition*, Software Engineering, IEEE Transactions on **30** (2004), pp. 311–327.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1291834