Resource Metrics for Service-Oriented Infrastructures

Dmytro Rud[1], Andreas Schmietendorf[1,2], and Reiner Dumke[1]

[1] Software Engineering Group, Department of Distributed Systems,
Faculty of Computer Science, Otto von Guericke University of Magdeburg,
Universitätsplatz 2, 39106 Magdeburg, Germany
{rud, schmiete, dumke}@ivs.cs.uni-magdeburg.de
[2] Berlin School of Economics, Faculty of Company-Linked Programmes,
Neue Bahnhofstr. 11-17, 10245 Berlin, Germany
schmiete@fhw-berlin.de

Abstract. Resource quality is one of the dimensions of software quality assessment and must be appropriately addressed in service-oriented architecture. In this paper some resource metrics for distributed systems that conform to the service-oriented concepts will be proposed. Similarities and differences between service-oriented, component-based and web-based software engineering approaches will be analysed in the context of involved resources and their quality impact.

1 Introduction

Software quality assessment is an important objective of software engineering and has big relevance in the context of SOA as well. The three dimensions of software quality are product, process and resources. In this paper we will present our view on significant run-time resource properties in service-oriented infrastructures.

Note that service-oriented architecture is an approach/an ideology, not a product. Therefore it is impossible to attribute neither resources nor resource metrics to it. Instead of that, the metrics introduced in this paper relate to distributed software systems built in accordance with SOA – i.e. implementations of the SOA principle. These distributed systems will be hereinafter referred to as "service-oriented infrastructures" or "service-oriented systems" (simply "systems" for short).

We will consider the following resources in service-oriented systems:

Network infrastructure. Its concrete logical topology can be point-to-point, bus (e.g. when a Enterprise Service Bus is used as an intermediate access layer), or some combination of them.

Service provider nodes. Each node is connected to the network and hosts a software infrastructure (application server(s)) that runs the services (service instances) and provides access to service endpoints and service metadata.

Services – pieces of software running on the nodes, whose functionalities and metadata are uniformly accessible for clients through the network.

Service functionalities (reactions to classes of incoming messages). Under RPC[3]-oriented interaction style this corresponds to operations' invocations, but the dominant interaction style today is document-oriented (i.e. asynchronous messaging), therefore the term "functionality" seems to be more correct here. Note also that some service design approaches consider every service to possess only one functionality (i.e. to provide a single operation when adhering to the RPC-oriented style). In this case the terms "service" and "service functionality" are synonymous.

There are also a few generalized metrics which relate to the system as a whole.

Due to dynamic nature of service-oriented systems, the boundary between product and resource, i.e. between design-time (static) and run-time (dynamic) properties, is somewhat diffused. For example, message sizes can be considered as either product or resource metrics.

Metrics proposed in this paper are intended to help answer the following questions:

- What is the current utilization of network and nodes?
- How much is it influenced by an invocation of a service's functionality?
- Are service versions managed well?
- What is the performance behavior of elementary and composite services?

The rest of the paper is organized as follows. The next section gives a review of related work. Section 3 discusses the question of the applicability of resource metrics from other distributed software architectures in the context of SOA. Our metrics are introduced in Section 4. Section 5 concludes the paper.

2 Related Work

Quality assessment and assurance of service-oriented infrastructures constitute an actual research topic. The product quality aspect, in particular service design guidelines, is well discussed in [1, 2]. We had presented a set of product metrics in [3]. As mentioned in the introduction, some product metrics can be relevant in the resource context as well, therefore some metrics from [3] will appear in this paper too. A formal quality model of single web services is described in [4], web services availability and performance problems are examined in [5].

Although there exist many tools on the market that monitor and analyse resource utilization in service-oriented systems, this subject is practically not in the least reflected in scientific literature. One of likely causes for this situation is the fact that there are many ways to implement SOA, and it is thus complicated to develop a resource quality model that would fit for all of them.

Explanation of resources' impact on performance and availability of web applications and client-server systems is given in [6]. Caching and XML processing

[3] Remote procedure call

issues are discussed in [7]. Author of [8] asserts that the rate of valid transaction completion should be considered as the key run-time performance metric for SOA environments.

Resource metrics are actually used by service management frameworks which provide functionalities like service performance monitoring or service matchmaking. These frameworks include, for example, WSLA [9], WSOI [10], WSMN [11], WS-QoS [12], and many other industrial and academical research projects [5].

3 SOA's Resemblances to Component- and Web-Based Applications

From the resource consumption point of view, service-oriented infrastructures share some properties with component-based [13, 14] and web-based [6] applications. Therefore arises the question of whether and to what extent is it possible to apply existing resource metrics from the component-based software engineering (CBSE) and web applications domains in the context of SOA. This applicability seems to be substantial, but the following conceptual differences between the respective approaches must be taken into account:

- Services can be composite (i.e. represent structured collaborations of other services, possibly with many "cascading" levels), while component-based and web-based applications do not support this technique as a rule.
- In CBSE it is impossible that many versions of a component are available simultaneously (in web engineering there is no version concept at all), but this can be the case in a service-oriented system.
- Unlike components, services involved in a transaction (in a composite service invocation, a business process, a workflow) can reside on different nodes in the network (thus possibly in different responsibility domains) and do not share single address space. Communication between services is thus unreliable and relatively slow, and the data transferring time cannot be neglected.
- Services represent functionalities and are generally stateless by design (like web resources, but unlike components)[4], therefore replication and load balancing can be arranged. Replication mechanisms give the possibility to recover a service collaboration in the case of partial failure.
- Unlike both component-based and web-based applications, service interactions can proceed asynchronously. Therefore performance metrics like response time are not always available.
- Non-functional run-time properties of services are often explicitly and formally (i.e. machine-readably) described in form of service or operational level agreements (SLA or OLA, respectively). Service providers are responsible for SLA fulfillment. This approach is rather uncommon in component-based and web-based environments.
- An especial resource in service-oriented systems is services' metadata.

[4] Some types of services, e.g. web services, can use session management mechanisms, but this approach is not widely adopted.

4 Introduction of the Metrics

In this section we describe our resource metrics for service-oriented infrastructures. The following three resource quality aspects will be considered in this connection: performance, service versioning, and reliability. Each aspect will be discussed in its own subsection.

4.1 Performance

The main performance characteristic of the network is its (current) throughput. It depends on the network topology. When a centralized bus like ESB is used, the throughput can be considered to be consistent in the whole system; in this case, the perfomance is determined for the most part not by throughputs between connected components and the ESB, but by the ESB's internal processing mechanisms like XSLT transformations, intermediate storage, security, transaction management, ans so on. However, if the ESB is used in an inter-enterprise environment, network latencies for the communication links between the bus and (external) components can have to be considered as well.

In the latter case, as well as in the absence of a centralized middleware, i.e. when every link between a pair of nodes has its own throughput value, network performance metrics relate to point-to-point throughput.

For these two variants we correspondingly introduce the following metrics:

CTY	– **C**onsistent **T**hroughput in the S**y**stem,
$T2N[n_1, n_2]$	– **T**hroughput between **2** **N**odes n_1 and n_2.

Unit of measurement for them is *bytes/sec*.

Performance of a node is first of all characterized by its (current) utilization, defined as fraction of time during which the node is busy with serving incoming requests. Other performance indicators are message rates and network traffic processed by the node. These properties are covered by the metrics:

$UN[n]$	– Overall **U**tilization of the **N**ode n,
$IMRN[n]$	– **I**ncoming **M**essage **R**ate of the **N**ode n (*messages/sec*),
$OMRN[n]$	– **O**utgoing **M**essage **R**ate of the **N**ode n (*messages/sec*),
$ITN[n]$	– **I**ncoming **T**raffic of the **N**ode n (*bytes/sec*), and
$OTN[n]$	– **O**utgoing **T**raffic of the **N**ode n (*bytes/sec*).

(The four latter metrics have been already introduced in [3].)

As mentioned above, utilization of nodes constitutes from fractions of time used to serve individual requests (to execute certain functionalities in response to incoming messages). Average sizes of such fractions ("processing costs" of average individual requests) can be considered as performance metrics on deep specification level. In the network throughput context, every service functionality is characterized by sizes of input and – in the case of RPC interaction style – output messages[5]. In order to be compatible with notational conventions used in our product metrics proposal [3], we will use here the term "operation" in the sense of "service functionality". Corresponding metrics are:

$AUO[m]$	– **A**verage **U**tilization (of the node that provides the corresponding service) caused by the **O**peration m (*seconds*),
$AIMSO[m]$	– **A**verage **I**nput **M**essage **S**ize for the **O**peration m (*bytes*),
$AOMSO[m]$	– **A**verage **O**utput **M**essage **S**ize for the **O**peration m (*bytes*).

If the service under consideration is composite, invocation of its functionalities imply a number of cascading calls to subordinate services. These calls cause additional utilization of corresponding nodes. To have an exact picture of the influence of an invocation upon utilizations of all N nodes of the system, the tuple

$$\langle AUO_1[m], AUO_2[m], \ldots, AUO_N[m] \rangle$$

should be analysed instead of the consideration of the single node. However it is obvious that the white-box view is necessary to obtain these value.

One of possible scenarios of SOA implementation is a system consisting of a set of service providers, a business process integrator and a set of clients of the latter, i.e. business process consumers. The mission of the integrator is to select an optimal set of third-party services, to orchestrate a composite service from them by filling out a business process description template with all information necessary to start the process – i.e. with partner links, addresses, etc., and finally to provide the latter to the customers.

The current and maximal possible numbers of simultaneously running business processes ("top-level transactions") can be important generalized indicators for the integrator. Therefore we introduce two corresponding metrics:

$ANBPY$	– **A**verage **N**umber of **B**usiness **P**rocesses in the S**y**stem,
$BPCY$	– **B**usiness **P**rocesses' **C**apacity of the S**y**stem.

To calculate the business processes' capacity of the system, deep analysis of the processes and their environment is necessary. An initial approach for such analysis was proposed in [15].

Other generalized metrics (already mentioned in [3]) give a "bird's-eye view" on the system's performance:

MRY	– Overall **M**essage **R**ate in the S**y**stem (*messages/sec*),
NTY	– Overall **N**etwork **T**raffic in the S**y**stem per one unit of time (*bytes/sec*).

[5] Fault messages should be taken into account as well

Note that we do not take into account possible use of caching mechanisms. Firstly, it is complicated to determine caching impact on resources utilization in technology-independent manner. Secondly, it is not very clear at all what the caching can mean under asynchronous document-oriented interaction style (as opposed to synchronous RPC-oriented style). For roughly the same reason we do not discuss scalability issues here.

4.2 Service Versioning

The possibility of different versions of the same service to be active simultaneously is a distinguishing feature of service-oriented systems. Version management has its quality aspects and thus must be addressed by software engineering in order to avoid negative consequences of poor versioning organization. In particular, a service with many short-lived versions can complicate both its maintenance and development of clients.

Two types of resources are appropriate in this context – there are installed service versions per se and services' metadata, i.e. formal (machine-readable) descriptions of services' functional and non-functional properties. A metadata repository (or registry) is an essential component of service-oriented infrastructures, serving for loose coupling and dynamic binding, and enabling agility in that way.

For individual services, the following versioning metrics may be relevant:

$CVS[s]$ – **C**ount of (simultaneously deployed) **V**ersions of the **S**ervice s,
$ALTVS[s]$ – **A**verage **L**ife **T**ime of **V**ersions of the **S**ervice s.

Their counterparts on the system level are:

$ACSVY$ – **A**verage **C**ount of **S**ervices' **V**ersions in the S**y**stem,
$ALTSVY$ – **A**verage **L**ife **T**ime of **S**ervices' **V**ersions in the S**y**stem.

In the absence of any versioning mechanisms, the only possible metric is

$MCFS[s]$ – **M**etadata **C**hange **F**requency of the **S**ervice s.

Service's metadata instability can break the work of existing clients and should be avoided. Proper versioning mechanisms should be used instead.

One of possible services' metadata types are service level agreements (SLAs), which have been already mentioned in this paper. The next subsection discusses SLA fulfillment issues and introduces a few SLA-related metrics.

4.3 Reliability

Service-level agreements are parts of service contracts and uniformly describe non-functional properties of the services. Sustained fulfillment of the SLA guarantees can be considered as the main quality criterion of a service.

Three conventional SLA fulfillment states can be distinguished:

Green area – All SLA conditions are consistently met,
Yellow area – Although SLA conditions are met, indicators (for example, some aggregated performance indices) come near to the prescribed threshold,
Red area – SOA conditions are not met.

On the basis of these areas we define the following metrics:

- $SLACS[s]$ – **SLA** **C**ompliance of the **S**ervice s, measured as the fraction of time during which all SLA fulfillment indicators of the service lie in green and/or yellow areas,
- $SLAVDS[s]$ – **SLA** **V**iolation **D**anger of the **S**ervice s, measured as

$$\frac{\textit{fraction of time in yellow area}}{\textit{fraction of time in green and yellow areas}}.$$

Faults (improper or missing reactions to incoming messages) are one of the most probable causes of SLA violations. Possible fault manifestations are:

- The service is unable to receive the incoming message,
- The service responds with a fault message,
- The service sends no response at all (for RPC-styled interaction),
- The non-fault response comes too late.

Alternatively to the SLA-related metrics described above, the following metrics can be used to describe faults which happen in the system, i.e. to address a narrower and more technical view on SLA fulfillment (and to avoid subjectivity caused by the choice of SLA fulfillment states' boundaries):

$FRO[m]$ – **F**ault **R**ate of the **O**peration m per one unit of time. This value can be calculated as $\frac{\textit{count of faults}}{\textit{count of received messages}}$, and
FRY – Overall **F**ault **R**ate in the S**y**stem.

Turning back to the business process integrator scenario (see Subsection 4.1), we can draw a parallel between these fault metrics and the rate of valid top-level transactions (composite business processes) completion metric proposed is [8]. Fault rates of the composed services can be considered as most important quality indicators of the business process integrator's work.

5 Conclusions

In this paper we have presented some resource metrics for service-oriented infrastructures. From the viewpoint of resource utilization, there are not so much differences between service-oriented, component-based and web-based applications, but their nevertheless exist, and we have tried to take them into account in our analysis.

Resource	**Performance metrics**	**Versioning metrics**	**Reliability metrics**
Network	CTY, T2N, MRY, NTY		
Service provider nodes	UN, ITN, OTN, IMRN, OMRN		
Services		CVS, ALTVS, MCFS	SLACS, SLAVDS
Service functionalities (operations)	AUO, AIMSO, AOMSO		FRO
System as a whole	ANBPY, BPCY	ACSVY, ALTSVY	FRY

Table 1. Classification of introduced metrics

Table 1 systematizes the proposed metrics and shows the correspondence between various quality aspects covered by the metrics and corresponding resources.

The proposed set of metrics is definitely not exhaustive, but it constitutes a basis for discussion and for subsequent work in this field.

The metrics are formulated in a technology-independent manner, specific technologies and measurement procedures are out of scope of the paper. Thereupon certain adjustment can become neccessary to make the metrics applicable in the context of concrete systems.

One of evident possible improvements of our resource quality model can lie in the consideration of the temporal aspect. For example, message rates and fault rates can be time-dependent. This peculiarity can be very important if we have to develop system performance models like the one in [15]. But standard measurement scales (nominal, ordinal, interval, ratio, and absolute) do not permit to use functions (e.g. statistical distributions) as metrics' values. The same applies to tuple-structured data as occurred in Subsection 4.1.

References

1. Artus, D.J.N.: SOA realization: Service design principles. IBM developerWorks (February 2006) http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design/.
2. Hess, A., Humm, B., Voß, M.: Regeln für serviceorientierte Architekturen hoher Qualität. Informatik Spektrum **29/6** (Dezember 2006) 395–411 ("Rules for service-oriented architectures of high quality", in German).
3. Rud, D., Schmietendorf, A., Dumke, R.: Product metrics for service-oriented infrastructures. In Abran, A., Bundschuh, M., Büren, G., Dumke, R., eds.: Applied Software Measurement. Proc. of the International Workshop on Software Metrics and DASMA Software Metrik Kongress (IWSM/MetriKon 2006). Magdeburger Schriften zum Empirischen Software Engineering, Potsdam, Germany, Hasso-Plattner-Institut, Shaker Verlag (November 2006) 161–174
4. Thielen, M.: Qualitätssicherung von Webservices. Entwurf eines allgemeinen Qualitätsmodells für eine Webservice-Zugriffsschicht. Master's thesis, Universität

Koblenz-Landau (2004) (“Quality assurance of web services. Development of a generic quality model for a web service access layer”, in German).

5. Rud, D.: Qualität von Web Services: Messung und Sicherung der Performance. VDM Verlag Dr. Müller, Saarbrücken (2006) (“Quality of web services: Measurement and assurance of performance”, in German).
6. Menascé, D.A., Almeida, V.A.F.: Capacity planning for web services: metrics, models, and methods. Prentice Hall (2002)
7. Cohen, F.: FastSOA: The way to use native XML technology to achieve service oriented architecture governance, scalability, and performance. Morgan Kaufmann Series in Data Management Systems. Elsevier Books, Oxford (January 2007)
8. Noel, J.: Transaction completion: The new performance metric for service oriented architecture environments. Technical report, Ptak, Noel & Associates (2005) http://www.ptaknoelassociates.com/content/library/2005/Certagon_transaction_completion.pdf.
9. Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Systems Journal **43**(1) (2004) 136–158 http://www.research.ibm.com/journal/sj/431/dan.pdf.
10. Tosic, V.: Service Offerings for XML Web Services and Their Management Applications. PhD thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada (August 2004) http://flash.lakeheadu.ca/∼vtosic/TosicThesis-Final.pdf.
11. Machiraju, V., Sahai, A., van Moorsel, A.: Web services management network: An overlay network for federated service management. Technical Report HPL-2002-234, HP Labs (2002) http://www.hpl.hp.com/techreports/2002/HPL-2002-234.pdf.
12. Tian, M.: QoS integration in Web services with the WS-QoS framework. PhD thesis, Fachbereich Mathematik u. Informatik, Freie Universität Berlin (November 2005) http://www.diss.fu-berlin.de/2005/326/index.html.
13. Gao, J.Z., Tsao, H.S.J., Wu, Y.: Testing and quality assurance for component-based software. Artech House (2003)
14. Szyperski, C.: Component software: Beyond object-oriented programming. Addison Wesley (1998)
15. Rud, D., Schmietendorf, A., Dumke, R.: Performance modeling of WS-BPEL-based web service compositions. In: Proc. of the IEEE Service Computing Workshops (SCC 2006), Los Alamitos, CA, USA (September 2006) 140–147