PEESOS: A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Luiz H. Nunes *, Luis H. V. Nakamura *, Bruno T. Kuehne *,
Edvard M. de Oliveira *, Rafael M. de O. Libardi *, Lucas J. Adami *,
Julio C. Estrella *, Stephan Reiff-Marganiec †
* *University of São Paulo (USP)*
Institute of Mathematics and Computer Science (ICMC), São Carlos-SP, Brazil
Email: {lhnunes, nakamura, btkuehne, edvard, mira, ljadami, jcezar}@icmc.usp.br
† *University of Leicester*
University Road, Leicester, LE1 7RH - UK
Email: srm13@le.ac.uk

***Abstract*—Performing functionality testing in service-oriented architectures is not a trivial task. The difficulty is especially the large number of components that may be present in a SOA such as brokers, providers, service registries, clients, monitoring tools, data storage tools, etc. Thus, in order to facilitate the process of conducting functional testing and capacity planning in service-oriented systems, we present PEESOS. This first version is a functional prototype that offers facilities to assist researchers and industry to test their new applications, allowing collaborations that can be done between the participants to achieve an appropriate objective when developing a new application. The first results show that it is possible to make a planning environment easier to operate and to readily obtain results for performance evaluation of a target architecture. Since this is a first version of the prototype, it has interface and scalability limitations as well as needing improvements in performance of the logs repository and also in a core engine. We hope that such limitations can be corrected in the near future, including gathering information from the scientific community to make the prototype a useful and accessible tool. PEESOS is on-line and available at http://peesos.wsarch.lasdpc.icmc.usp.br.**

***Keywords*-Web Services, Service Oriented Architecture, Quality of Service, Performance Evaluation, Capacity Planning, Functional Tests**

I. Introduction

SOA is an architecture which provides applications as services. It improves the systems interoperability and provides interfaces for legacy systems. However, enterprises adopting SOA should make efforts to perform as many tests as possible to ensure minimum levels of performance. [1].

SOA tests can be classified into four categories: regression, integration, non-functional and functional testing. Functional tests aims to validate the functional properties of a service. Regression tests reuses previous test cases, when modifications occurs in an existing system to ensure that all features are still working. Integration tests aims to make several components of a system work together as expected. Non-functional tests aim to ensure that non-functional properties reach appropriate values during the execution [3].

Functional tests can be divided into other categories: unit, composition and end-to-end. Unit tests validate that a service meets its functional requirements. Composition test aim to validate workflows or individual services which are part of a composite service. End-to-end tests test all business flows that constitute a individual or composite service. Basically to perform end-to-end testing of a SOA we need: 1) Identification of business critical test scenarios, 2) Identification of Dynamic Composition Sequences, 3) Establishment of realistic test environments and data and 4) Report on and Analyse Test Results [9].

Capacity planning is a common strategy used to measure the capacity of traditional software systems. It is difficult, expensive and hard to adequately plan the capacity for SOA end-to-end applications. The major problems of capacity planning can be summarized as lack of tools and skills, integration of SOA components, tests across organizational boundaries and security requirements [2].

In this paper, we propose a tool called Planning and Execution for Experiments in Service Oriented Systems (PEESOS). PEESOS aims to perform capacity planning tests for end-to-end SOA applications in real and simulated SOA systems. PEESOS differs from others approaches in at least two points: it guides the user through a model based on a set of commons SOA entries and it allows to establish a full factorial design experiment. PEESOS also provides a collaborative workload generator based on a Client-Server model to establish a realistic load test environment for capacity planning test. Compared to the tools presented in the related work, the key point of PEESOS is that it is available online to be evaluated.

The paper is organized as follows: Section II presents a literature review of existing approaches in SOA testing tools. Section III describes the proposed tools structure and how it works. Section IV describes the methodology and configurations used for the experiments. The results are then discussed in Section V. Finally, the conclusions and directions for future work are presented in Section VI.

II. Related Work

Experiments in SOA pose three main problems: lack of an intuitive execution environment; integration of available resources; and performance monitoring during tests [12].

Some tools can create virtual environments to execute automated tests in SOA based on WSDL files [1][2][3]. They can perform functional, regression, compliance and load tests. However they are commercial, closed-source and usually prohibitively expensive for small groups.

In [13] the TESSI (TESting of Service Implementations) tool is presented as part of the TASSA framework to achieve web service environment testing. TESSI can generate SOAP request templates based on HTTP, SOAP and BPEL variable levels and define unitary tests. It can also support data driven testing, test management and test execution. However, the tester needs to specify value ranges for each variable [9]. Other work like WSDLTest [16], monadWS [18] and BPELUnit [11] can also automate tests based on web services description files.

Brebner [2] presents the SOPM (Service-Oriented Performance Modeling) framework which is a visual tool to simulate a service-oriented system based on services and their composition. Although the error was up to 15 percent in a functional test, this tool helps to understand the scalability and performance before service deployment.

Gu and Ge [6] shows a method to auto-generate performance test cases based on two consecutive steps. The first step is the work-flow analysis, where QoS factors are identified. The second step is a genetic algorithm that creates the test cases.

Smit and Stroulia [15] presents the SASF (Services-Aware Simulation Framework) to create executable simulations based on existing data about services. The SASF focus on capacity planning and enable users to interact with a running simulation through an user interface and an API. It also includes a flexible metrics-gathering component that can be extended to collect and visualize new metrics.

Xing and Yao[17] proposes a system that performs collaborative experiments in SOA. Internet users are able to communicate and collaborate with service requests for large-scale experiments through that tool.

Smit and Stroulia [15] also presents a simulation frameworks survey for SOA and concluded that service-composition testing before real deployment is the most common task supported by the analyzed frameworks. The Table I shows a survey summary of the frameworks analyzed. WSDL is used to model most of the tools, except SOPM and our tool. Also, all non-commercial frameworks (excluding Testing Tools) use some kind of proof-of-concept evaluation method. Our tool (PEESOS) differs from the others frameworks because it enables evaluation in real environments.

Based on [17] this work proposes a tool for capacity planning tests in SOA within a real environment. These tests differ from [13] SOAP request templates by using a model based on a set of commons SOA entries to guide the user when configuring the tests. Also, this research complements [2] and [15] by adding a real workload environment to obtain more accurate metrics.

III. PEESOS

A. Overview

The PEESOS tool is a prototype which supports the modeling of design of experiments (DOE) in SOA. PEESOS also enables the execution of the modelled experiment in the target environment. The main purpose of PEESOS is to capture the results from the executions to predict the performance of the target system, under different resource configurations and workloads. It is also capable to predict non-functional metric such as QoS for clients in different environments. The main advantage from a prototype approach instead of simulation and analytic model is the possibility to have more accuracy of the results that help to detect bottlenecks which is not possible in simulation or by using analytical models.

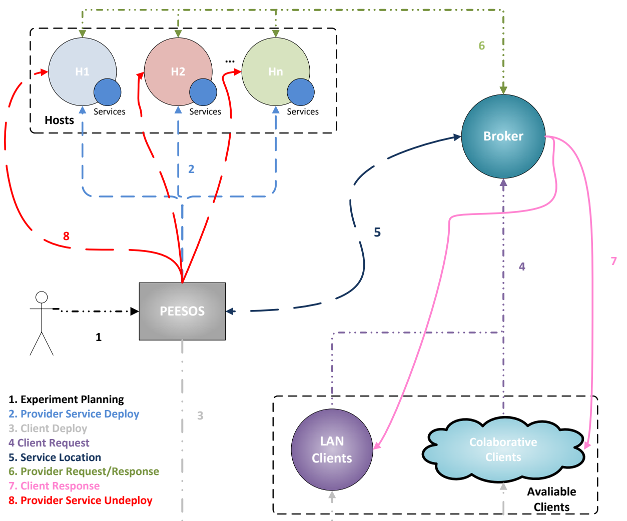


Figure 1: Workflow of tool

The DOE of this tools is based on a set of commons SOA entries, where test cases are generated through a model which describes the expected behaviour of an experiment. Each one of this entries affects the response variable and are called factors. The values which a factor can assume are

[1]SoapUI - http://www.soapui.org/
[2]GH Tester - http://www.greenhat.com/ghtester/
[3]CloudPort - http://www.crosschecknet.com/products/cloudport.php

Table I: Objectives, Modeling and Evaluation Properties. Adapted from [15]

	Objective	**Modeling**	**Evaluation**
SOPM	Verify perfomance and scalability pre-deployment	Visual, Jmeter for performance	Proof-of-concept (functional test)
Testing Tools	Composition and load testing automatically from WSDLs	WSDL	-
SASF	Create a virtual model of a componentized software application Focused on capacity planning	WSDL + perf. Profiles	Modeled two applications; statistically validated metrics
TESSI	provide test case generation, execution and management	WSDL, BPEL	Simple proof-of-concept
PEESOS	Provide end-to-end test case generation and execution focused on capacity	model with a set of SOA commoms entries	Proof-of-concept in virtual/real environment

called levels. Test cases generated by this model uses a full factorial design with every possible combination at all levels of all factors [8] being covered.

The aim of this work is to reduce the complexity in performing capacity testing in SOA by abstracting the steps of service deployment and to generate load for these experiments. The PEESOS tool performs all steps to deploy a service in a real or simulated environment. After the deployment of a service, a synthetic workload is generated in a collaborative form which allows to obtain more accurate QoS parameters.

Figure 1 shows the sequence of steps when using the PEESOS tool:

1) the stakeholder prepares the experiment parameters: number of hosts, number of clients, service, client application, workload, etc.
2) services are deployed in the selected hosts .
3) the client application is deployed to selected clients.
4) the clients perform requests to the broker.
5) the broker chooses a host to attend the client request.
6) the broker forwards the client request to the selected provider.
7) the broker forwards the host response to the client.
8) the services are undeployed from the hosts.

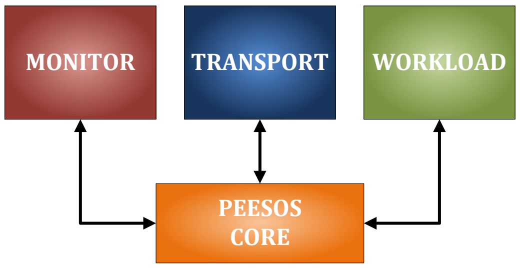


Figure 2: Modules of PEESOS tool

Figure 2 shows the overall structure of the tool. The tool is composed of four modules, described in more detail in the next sections. Briefly:

Monitor: monitors the status of providers and clients available for experiments;

Transport: manages the file transfer and command execution of clients and service providers;

Workload: manage the distribution of workload among the clients requests;

PEESOS Core: manages the communication between the other three modules and contains the main structures of the tool.

B. *Monitor*

The *Monitor Module* has a set of methods to monitor the availability and condition of services providers and client devices. The available information is monitored in the service providers by the Ganglia Monitoring System[4]. All information collected from the Ganglia Monitor is stored in a relational database. The location and availability of the providers are retrieved from client devices exclusively through a client application that uses a socket based client-server model.

C. *Transport*

The *Transport Module* has a set of methods whose purpose is the transfer of data to clients and service providers. Service providers can receive data trough the SSH protocol or sockets. Data transferred to service providers contains the service that will be deployed. Like the *Monitor Module*, client devices can only received data through a client application. Data transfer through this can contain the client application that will send requests to providers or a command to execute a client application.

D. *Workload*

The *Workload Module* has a set of methods whose purpose is to generate a synthetic workload model that will be used to control clients requests. These models use probability distributions to generate workload data.

[4]http://ganglia.sourceforge.net/

E. *PEESOS Core*

The *PEESOS Core* is the main module, it has a set of methods to orchestrate the experiment planning and execution. The PEESOS Core is also responsible for performing the communication and synchronization between the other modules. It communicates with the *Monitor Module* to determine which provider and clients are available for experiment planning. It knows which providers and clients are selected by the stakeholder for communication with the *Transport Module* and transfers all needed data. The *Workload Module* is accessed when the interval between each request has to be determined and performs requests trough the *Transport Module*. The set of common SOA entries has an exclusive class in this module with basic attributes to perform an experiment. If a specific experiment is not defined with by common entries, this class can be extended and adapted to accomplish it.

IV. Performance Evaluation

In this section, we present a performance evaluation of PEESOS in a functional case study which is related to SOA components and web services. The motivating example for our evaluation is a SOA prototype named WSARCH[5] running a synthetic web service.

The main aim of the WSARCH architecture is to provide a basic infrastructure to detail and solve problems related to workload, service composition, fault tolerance, network and security of messages and components [5].

The synthetic web service is a CPU bound service, developed to calculate the factorial of a number in an iterative way based on formula, $n! = \prod_{k=1}^{n} k, \forall k \in \mathbb{R}$. This service was chosen because it can easily stress the SOA target by generating large messages.

A. *Experiment Environment*

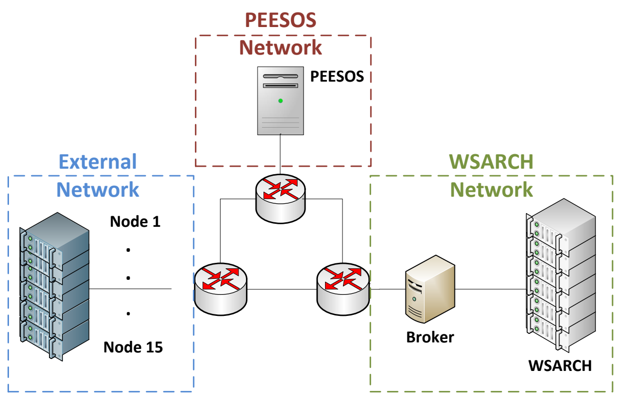


Figure 3: Test Environment

[5]http://wsarch.lasdpc.icmc.usp.br/

The test environment for experiments is prepared with a set of machines disposed in an external network to perform requests. The WSARCH Broker is available to deploy applications and handle requests. Figure 3 shows the test environment arrangement. PEESOS manages external machines to make requests to the WSARCH broker, which is responsible for scheduling those requests among providers and then responds to them. PEESOS also uses the WSARCH architecture to deploy services in providers and optionally register them in a UDDI repository.

The test environment is composed of one virtual machine to execute PEESOS hosted in a physical machine and fifteen hosts as available clients. Table II describes these machine configurations. The WSARCH architecture is composed of one broker, one LogServer, twelve virtual providers (four virtual machines per host) and twelve virtual UDDIs (four virtual machines per host). All information regarding access to Providers, UDDIs, Broker and LogServer can be found at http://wsarch.lasdpc.icmc.usp.br/infrastructure.

Table II: Clients and PEESOS Configuration

	Clients	**Hosts**	**PEESOS**
Processor	AMD Processor Vishera 4.2 Ghz		2 Virtual Processor
Memory	32 GB RAM DDR3 Corsair Vegeance		2GB RAM
Hard Disk	HD 2TB Seagate Sata III 7200RPM		50GB
Operating System	Linux Ubuntu Server 12.04 6u Bits LTS		Linux Ubuntu Server 12.04 6u Bits LTS
Applications	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0	Qemu / KVM	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0

1) Target Architeture: The WSARCH architecture will briefly be presented to allow a better understanding of the test system. WSARCH has five distinct modules: the client application, the providers, the Broker, the UDDI *Universal Description, Discovery and Integration* registry and the Log Server. Figure 4 shows the relationships between these components.

Clients perform requests with QoS parameters to the *Broker*. The *Broker* is responsible for finding the specific service to meet the request, which will be available in one of the providers [7]. Providers are repositories of services and work closely with a core group responsible for processing the messages of requests and responses, called *Apache Axis 2* [5]. The location and information of service providers are stored in a UDDI registry, which was modified to contain qualifications (QoS) and characteristics of services.

The architecture also has the *Log Server*, which is a database responsible for storing all data transactions between components. Besides, information of quality of service offered by providers are updated every second, collected by a Ganglia monitor [10] and transmitted from one module to another under *Broker* management. Table III describes the configuration of the WSARCH components.

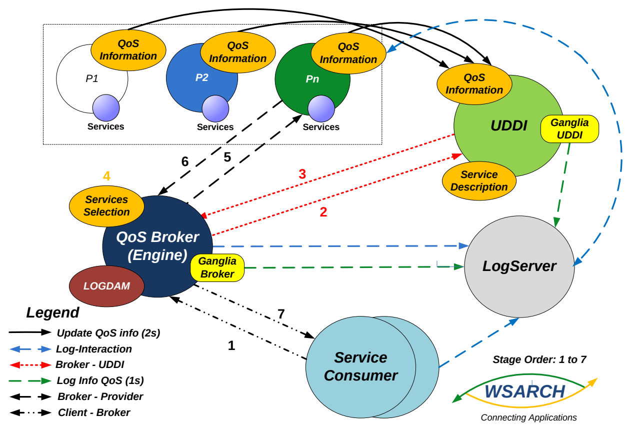


Figure 4: *Web Service Architecture* - WSARCH [5]

Table III: WSARCH Architeture Configuration

	WSARCH Environment		
	Broker and LogServer	**Hosts**	**Virtual Machines (Provider and UDDI)**
Processor	AMD Processor Vishera 4.2 Ghz		2 Virtual Processor
Memory	32 GB RAM DDR3 Corsair Vegeance		2GB RAM
Hard Disk	HD 2TB Seagate Sata III 7200RPM		50GB
Operating System	Linux Ubuntu Server 12.04 6u Bits LTS		Linux Ubuntu Server 12.04 6u Bits LTS
Applications	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0	Qemu/KVM	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0

WSARCH has a standard selector named *Default Selector*, which works directly in the *Broker*. This selector takes QoS values as parameter for provider selection and prevents them being overloaded. These QoS values are updated from time to time by the service providers.

B. Experiment Design

We used the model available in PEESOS tool to design the test experiment. Relevant parameters such as number of providers, number of clients, client locations, workload type, application parameters and number of replications were defined to gather as much information as possible to compare the performance in server and client sides.

Table IV shows the experiment design values. A full factorial experiment is performed based on diferents levels of four factors: number of providers (1 and 12), number of clients (8 and 15), workload type (exponential and uniform) and factorial number (1.000 and 3.000), totaling a set of sixteen experiments. These experiments are divided into four subsets, each one with four experiments. All experiments are replicated ten times to get a 95% confidence interval. Besides, each experiment is performed with a total of one thousand and two hundred requests, where the total number of requests is divided by the total number of clients. Also, the times to find the service in the UDDI repository were

Table IV: Experiment Design

	Providers	Clients	Workload	Factorial	Subset
Exp1	1	15	Exponential	1.000	1
Exp2	1	15	Exponential	3.000	
Exp3	1	15	Uniform	1.000	
Exp4	1	15	Uniform	3.000	
Exp5	1	8	Exponential	1.000	2
Exp6	1	8	Exponential	3.000	
Exp7	1	8	Uniform	1.000	
Exp8	1	8	Uniform	3.000	
Exp9	12	15	Exponential	1.000	3
Exp10	12	15	Exponential	3.000	
Exp11	12	15	Uniform	1.000	
Exp12	12	15	Uniform	3.000	
Exp13	12	8	Exponential	1.000	4
Exp14	12	8	Exponential	3.000	
Exp15	12	8	Uniform	1.000	
Exp16	12	8	Uniform	3.000	

subtracted from the total response time to allow for analysis to focus on execution time of services.

The Exponential Workload was generated based on Equation 1 [14],

$$f(x,\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{for } x \geq 0, \quad (1a) \\ 0 & \text{for } x < 0. \quad (1b) \end{cases}$$

where the mean of exponential distribution is represented by $\lambda = 9000, \forall x \in [0, \infty)$. Uniform Workload was generated based on the Equation 2 [4],

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{for } a \leq x \leq b, \quad (2a) \\ 0 & \text{for } x < a \text{ or } x > b. \quad (2b) \end{cases}$$

where the minimum value represented by $a = 1000$ and the maximum value represented by $b = 21000$. The requests interval are defined based on random values from cumulative distribution function of Equations 1, 2. The values of λ in Equation 1 and a and b in Equation 2 were set to guarantee the maximum time interval closer between each other.

C. PEESOS Interface Configuration

The experiment configuration followed the interface model fluxogram, shown in Figure 5. To ease the understanding, we will show the configuration of Expr1 along with the explanation of the tool configuration fluxogram.

Firstly, in the "Experiment Configuration" phase, we need to input the *Number of Clients* (15), *Number of Providers* (1), *Number of Replications* (10), *Number of Requests* (80), *Service (Factorial) and Client (broker-client)* application and the *Interval* between experiments (10 min) like in Figure 6.

Then, we need to input the service parameters (1000-3000) in the "Service and Client App. Configuration" phase. After, we go to the "Clients Configuration" step, where we will select the Clients IPs (192.168.0.{20, 30, ... , 170, 180}) and then we need to configure the providers (port 2012). After these phases we go to the "Experiment Review" phase,

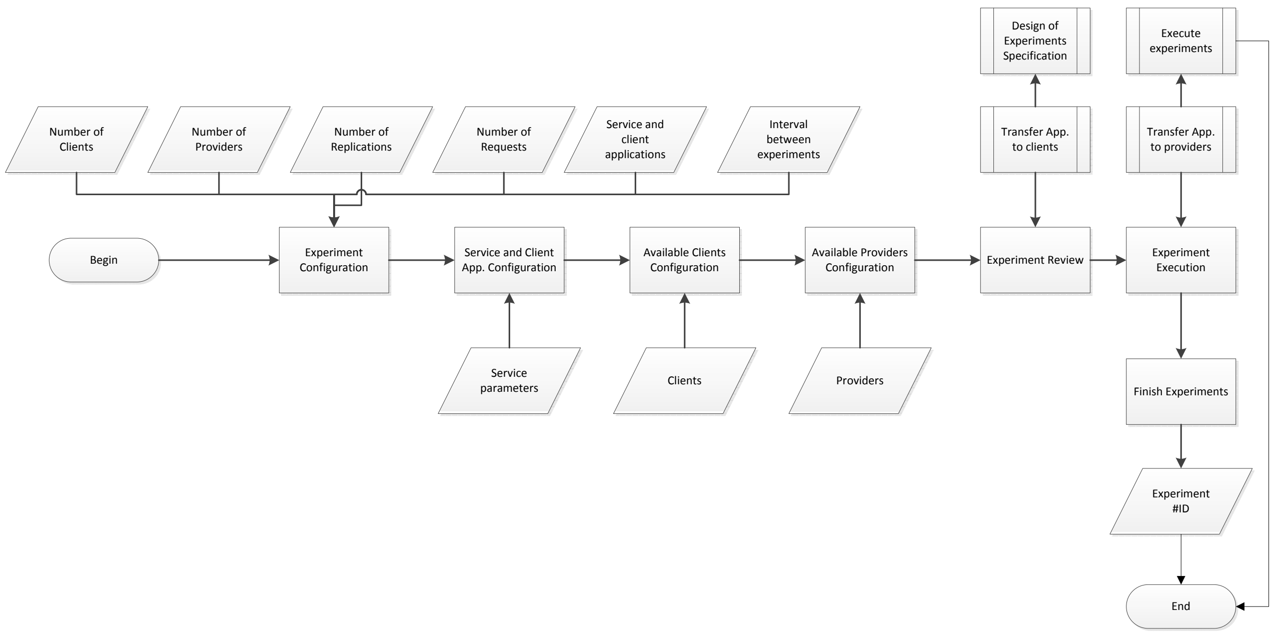
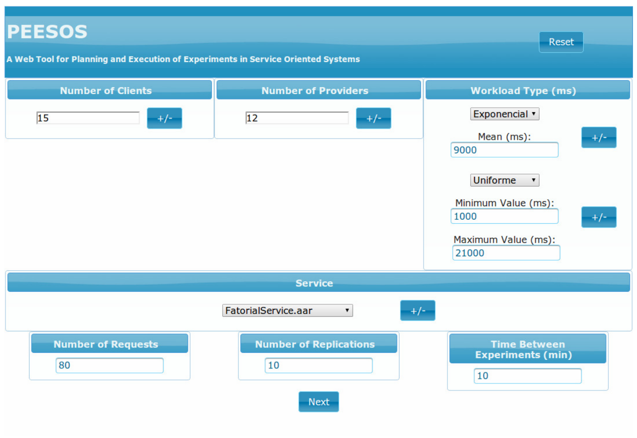


Figure 5: PEESOS Interface Fluxogram


PEESOS
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Number of Clients	15
Number of Providers	12
Workload Type (ms)	Exponencial
Mean (ms):	9000
	Uniforme
Minimum Value (ms):	1000
Maximum Value (ms):	21000
Service	FatorialService.aar
Number of Requests	80
Number of Replications	10
Time Between Experiments (min)	10

Figure 6: PEESOS Setup

where the App will be transferred to the clients and the "Design of Experiments Specification" is created. Finally, the experiment is executed in the "Experiment Execution" phase, where the Apps will be transferred to the providers and the experiment will be executed. Then we move to the "Finish Experiments" phase, where an unique *Experiment ID* will be generated to retrieve the experiments results later.

V. Results

In this section are reported the results obtained by the execution of the experiments described in Section IV. The gathered data enabled the analysis of the WSARCH execution chain and identify problems using UDDI repository.

A. *Results without WSARCH UDDI*

The results generated by the PEESOS tool execution without WSARCH UDDI are shown in Figure 7 which shows the interval plot and a square which indicates the mean in seconds. Analysing the results it is possible to notice that experiments with 15 clients took longer time to complete than those experiments with 8 clients. The factorial calculation of 1000 takes less time than factorial calculation of 3000.

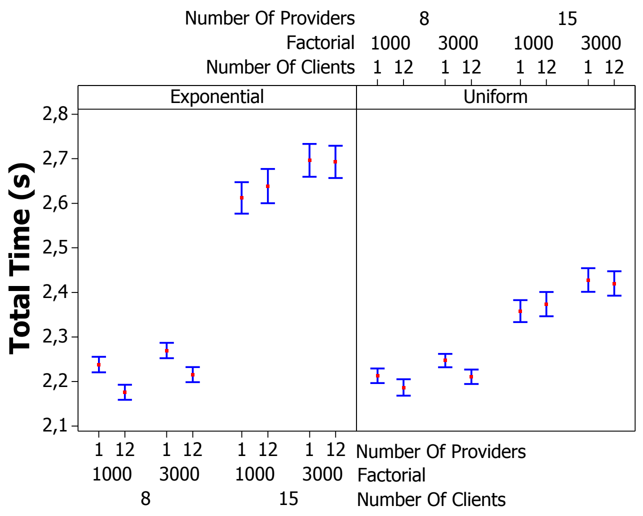


Figure 7: Interval plot graphic without UDDI search time

In a deeper analysis, we noticed that the exponential distribution using 15 simultaneous clients causes more overhead than the uniform distribution with any other combination of factors. With 8 simultaneous clients and using just 1 provider, the exponential distribution has a slightly higher average than uniform distribution for both factorial values. When we used 12 providers, the time results for both distributions were statically equals using the factorial calculations for 1000 and 3000.

Using 12 providers, the total time of request was better than 1 provider for 8 clients. When 15 clients are used, the time for both cases are statistically equivalent. This behavior is explained by the selection algorithm used in WSARCH, which does not have an update interval consistent with the rate of arrival of requests.

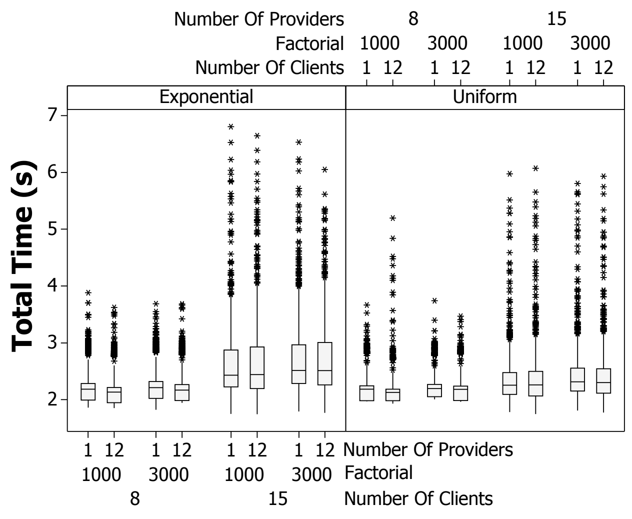


Figure 8: Box plot graphic without UDDI search time

Figure 8 shows the box plot based on the 1200 requests performed. The box plot makes it possible to identify the time periods of warm up, outliers and how the requests behave. The major part of requests which took longer than 4 seconds, occurs during the warm up period (possible caching and JIT - *Just In Time* compilation effects). The experiments with 15 clients had more outliers than the experiments with 8 clients. It happens because experiments with 15 clients the simultaneous requests increases and, consequently, the provider side has to handle a longer queue of requests. The box of each experiment has the same characteristics as Figure 7.

B. *Results with WSARCH UDDI*

The results generated by the PEESOS tool execution with WSARCH UDDI are shown in Figure 9, the interval plot and a square indicate the mean in seconds. The overall behaviour of this experiment is similar to the one presented in Figure 7. The experiments with 15 clients took longer time to complete than the experiments with 8 clients. It is important to highlight the results behaviour of 12 providers, which the total response time were higher than in the result of 1 provider. This seems very surprising, but when we consider the WSARCH UDDI time, an additional overhead occurs proportionally to the amount of records in the repository.

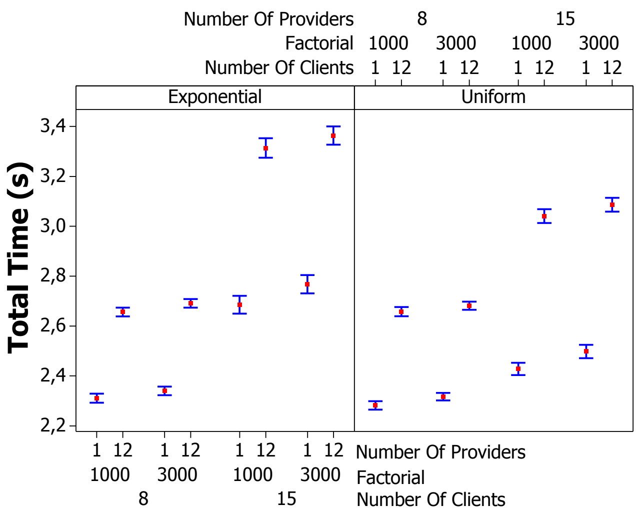


Figure 9: Interval plot graphic with UDDI search time

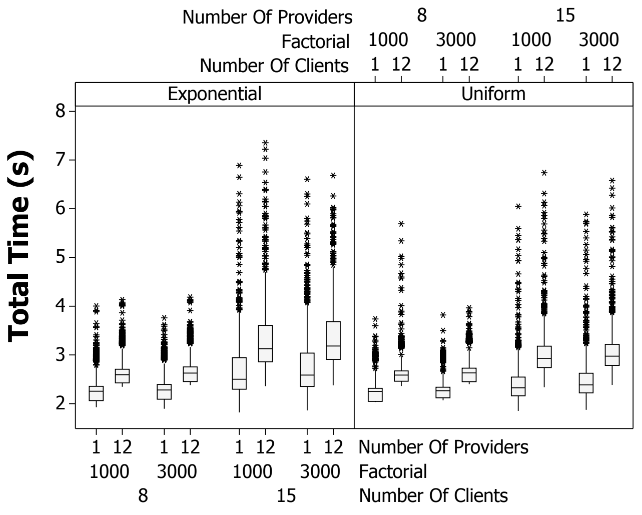


Figure 10: Box plot graphic with UDDI search time

In Figure 10 the box plot presents the results from an experiment where 1200 requests were performed. Similar to the experiment presented in Figure 8, in the warm up period, the major part of requests took longer than 4 seconds for 1 provider and 5.5 seconds for 12 providers. As in Figure 9, 12 providers have the box and outliers higher than experiments with 1 provider.

Analysing these results, we concluded that the increasing number of UDDI registered providers is a bottleneck in the WSARCH architecture. During the experiment others problems were identified, for instance, the Tomcats applications on the providers side were crashing as the log file got too large.

VI. Conclusion

In this paper the PEESOS tool was presented, the main goal of this tool is to facilitate the planning and execution of experiments in SOA. Capacity planning is used in PEESOS to observe the behavior of these architectures, with different types of workload. By using PEESOS, the bottlenecks can be detected as we demonstrated in Section V. Furthermore, the use of a real collaborative environment consider factors, which may influence specific attributes of a request that are not included in a simulated experiment, for example, network traffic congestion, routing, software configuration, etc.

PEESOS improvements such as a new interface, increasing the availability of the tool to be used by many users in the same time, and mechanisms for fault tolerance in the server and client side will be addressed in the future. The prototype presented in this paper and wizard guide is available in http://peesos.wsarch.lasdpc.icmc.usp.br/ and the experiments presented in this paper can be repeated according to Section IV-C.

VII. Acknowledgements

This project was financially supported by National Counsel of Technological and Scientific Development (CNPQ), process number 133841/2012-0 and Sao Paulo Research Foundation (FAPESP), process number 11/09524-7 and 13/26420-6.

References

[1] M. Bozkurt, M. Harman, and Y. Hassoun. Testing and verification in service-oriented architecture: A survey. *Software Testing Verification and Reliability*, 23(4):261–313, 2013.

[2] P. Brebner. Service-oriented performance modeling the mule enterprise service bus (esb) loan broker application. *Conference Proceedings of the EUROMICRO*, pages 404–411, 2009.

[3] G. Canfora and M. Penta. Service-oriented architectures testing: A survey. In A. Lucia and F. Ferrucci, editors, *Software Engineering*, volume 5413 of *Lecture Notes in Computer Science*, pages 78–105. Springer Berlin Heidelberg, 2009.

[4] G. Casella and R. Berger. *Statistical inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002.

[5] J. C. Estrella, R. H. C. Santana, and M. J. Santana. *WSARCH: An Architecture for Web Services Provisioning with QoS Support: Performance Challenges*. VDM Verlag Dr. Mller, 2011.

[6] Y. Gu and Y. Ge. Search-based performance testing of applications with composite services. In *Web Information Systems and Mining, 2009. WISM 2009. International Conference on*, pages 320–324, 2009.

[7] P. Harshavardhanan, J. Akilandeswari, and R. Sarathkumar. Dynamic web services discovery and selection using qos-broker architecture. In *Computer Communication and Informatics (ICCCI), 2012 International Conference on*, pages 1 –5, jan. 2012.

[8] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 1991.

[9] P. Kalamegam and Z. Godandapani. A survey on testing soa built using web services. *International Journal of Software Engineering and its Applications*, 6(4):91–104, 2012.

[10] M. Massie, B. Chun, and D. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[11] P. Mayer and D. Lübke. Towards a bpel unit testing framework. In *Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications*, TAV-WEB '06, pages 33–42, New York, NY, USA, 2006. ACM.

[12] D. Meirong and C. Shaoming. Design and realization of soa based management system for open experiment lab. In *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, pages 3182 –3185, april 2012.

[13] D. Petrova-Antonova, S. Ilieva, and V. Stoyanova. Tessi: A web service testing tool: Demonstration paper. In *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, pages 1–2, 2013.

[14] S. Ross. *Introduction to Probability and Statistics for Engineers and Scientists*. Elsevier Science, 2009.

[15] M. Smit and E. Stroulia. Simulating service-oriented systems: A survey and the services-aware simulation framework. *IEEE Transactions on Services Computing*, 99(PrePrints), 2012.

[16] H. Sneed and S. Huang. Wsdltest - a tool for testing web services. In *Web Site Evolution, 2006. WSE '06. Eighth IEEE International Symposium on*, pages 14–21, 2006.

[17] Y. Xing and E. Yao. Remote collaborative experiments based on service-oriented architecture(soa). In *Signal Processing Systems (ICSPS), 2010 2nd International Conference on*, volume 2, pages V2–605 –V2–608, july 2010.

[18] Y. Zhang, W. Fu, and C. Nie. monadws: A monad-based testing tool for web services. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 111–112, New York, NY, USA, 2011. ACM.