Clients Load Generator and Simulator

Svetlana Slavova
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
sds797@mail.usask.ca

ABSTRACT

A tool, Clients Load Generator and Simulator, is presented in this paper as a way for creating consumers' load and for simulating the consumers' behavior. The application generates requests in XML format and sends them to the desired endpoints. The target components are Web Services, accessible via their service descriptions. The tool is designed in a way to represent real clients' behavior – the calls within a client are synchronized, whereas between multiple clients the requests are executed in parallel. The time between the service calls is controlled by a time constant that is set for each request. In addition, the tool keeps logs of all results obtained during the simulation period. The application and the target services are tested in a single machine as well as in a distributed environment using RPC communication style (Apache Axis Framework, version 1.x) and Document communication style (Apache Axis Framework, version 2.0). The obtained results confirm that the proposed application can be used in experiments requiring invocation of Web Services. Furthermore, they show some differences in the response times of the services when using Axis 1 and Axis 2.

1. INTRODUCTION

The Clients Load Generator and Simulator for Web Services is a Java application for creating and executing clients' behavior. The workload is designed in XML format which allows the generation of the XML files to be done automatically by the program or manually by the user. Each request consists of the following information: *Unique identifier* (client id & request id); *URL* – endpoint to the desired Web Service; *Method name* – name of the method of the Web Service that has to be invoked; *Arguments* – arguments of the method; only String data types are taken into account; *Time* – time in milliseconds showing the relative time for sending the request after getting the response of the previous call.

The tool has three main functions (Figure 1):

- *To generate load for one client*. The calls represent the behavior of one client (1 Client ←→ Multiple Web Services). The user fills in the fields (URL, method name, arguments, and time) for each call and gets as a result an XML file containing the full behavior of the client. The application checks if all required fields are filled in. If not, an user error will occur;
- *To generate load for multiple clients*. The calls represent the behavior of multiple clients (Multiple Clients ←→ Multiple Web Services). The user fills in the names of the XML files that should be loaded. The application randomizes the single client's load and creates an XML file containing behavior for multiple clients;
- *To simulate the clients' behavior*. The application loads the XML file that contains multiple clients' load, parses the file in order to get the necessary information (client ids, request ids, URLs, method names, arguments, and times), and starts the execution of each client. The client is started in a separate thread. The threads invoke the Web Services, get the results, and write them into a text file (log-file).

  The application is able to request Web Services on both frameworks – Axis 1.x and Axis 2. When starting the simulation, the user can specify the communication style – RPC or Document.

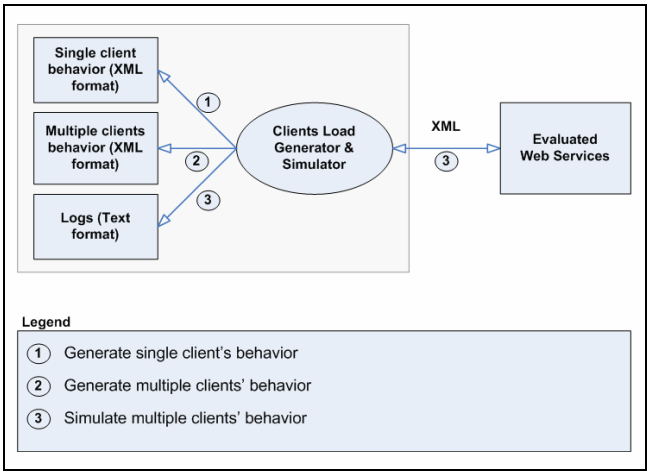

Figure 1. Clients Load Generator and Simulator for Web Services

The Clients Load Generator and Simulator deals with two types of requests:

- *Requests within a client requiring synchronization.* The behavior of the client consists of one or more steps (requests). When the result of the first request is obtained, the second call is sent to the next Web Service. Depending on the scheduled time for

execution, the request might wait for some milliseconds. Each client is executed in a different thread. Each thread starts timers for execution of the Web Services;

- *Parallel requests of different clients.* Since each client runs in a separate thread, the behavior of the different consumers is independent from one to another, which allows clients to send calls simultaneously to the same Web Services.

The remainder of the paper is structured as follows: Section 2 provides a problem definition; Section 3 describes the implementation of the tool – data structures, clients' requests representation, hierarchy of the implemented classes, and user interface; The section ends with some directions about how to compile and to run the application; Section 4 discusses the experiment setups and the results are presented in section 5; Finally, section 6 shows some conclusions and future work.

2. PROBLEM DEFINITION

The goal of the Clients Load Generator and Simulator Tool is to be helpful during Web Services experiments that require service invocations. The main questions of the paper are the following:

- How to make the tool similar to the real-world clients' behavior?
- Which is the minimum required information that should be obtained from the user during the generation of the clients' load as well as during the simulation period?

3. IMPLEMENTATION

The following programming mechanisms, languages, and techniques are used during the development of the application: *File I/O*, *Threads*, *XML file creation*, *XML file parsing*, *Web Services*, *Axis 1*, *Axis2*, *Java*, and *Java Swing*.

3.1. Data Structures

The clients' load is represented in XML format which makes it language- and platform-independent. The root element of the multiple clients' behavior, called *multiple_load*, wraps the whole XML document. Each client has a unique identifier as element of tag *client_id*. The requests of each consumer are unique as well and are defined as elements of tag *request_id*. They have a prefix *number_* and suffix – the number of the request. The parameters of each call can be found within tag *request*. They are: URL of a Web Service, method name, arguments of the method, time showing in how many milliseconds after getting the result of the previous request, the current request should be executed. Since the arguments of the methods could be multiple, they are wrapped in tag *arguments*. An example of generated multiple clients' load can be seen on Figure 2.

Example: Two clients with one request

```
<multiple_load>
    <client id="Svetlana1">
        <request id="number_1">
            <url>
                http://varna.usask.ca:8080/axis/Tests/test2.jws?wsdl
            </url>
            <method>show2</method>
            <arguments>
                <argument>Slavova</argument>
            </arguments>
            <time>3000</time>
        </request>
    </client>
    <client id="Svetlana2">
        <request id="number_1">
            <url>
                http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl
            </url>
            <method>show</method>
            <arguments>
                <argument>Svetlana</argument>
            </arguments>
            <time>5000</time>
        </request>
    </client>
</multiple_load>
```

Figure 2. Generated multiple clients' load

3.2. Clients' Requests Representation

In order to represent the requests of the clients, hash tables are used as data types. There are three hash tables that correspond to the structure of the data – *Clients*, *Requests*, and *RequestParams*. Table *Clients* consists of all clients' ids that participate in the generated XML file. There is one table *Requests* that corresponds to each client id. The keys of table *Requests* are the ids of the requests of the client. The values of the table are *RequestParams* hash tables. Each RequestParams table has four keys – *url*, *method*, *arguments*, and *time*. The structure of the clients' requests representation can be followed on Figure 3.

3.3. Hierarchy of the Implemented Classes

The implementation of the Clients Load Generator & Simulator is developed in Java. The application is structured in two packages: *GUI* and *Actions*. The *GUI* package contains two classes *UserInterface.java* and *SubFrames.java* that are responsible to provide all necessary methods for the interface and the interaction with the users. The *Actions* package provides classes that manage hash tables (*ManageHashtables.java*), parse XML files (*ParseXMlFile.java*), generate clients' load (*GenerateLoad.java*), and simulate clients' behavior (*SimulateClient.java*). The hierarchy of the classes is shown on Figure 4.

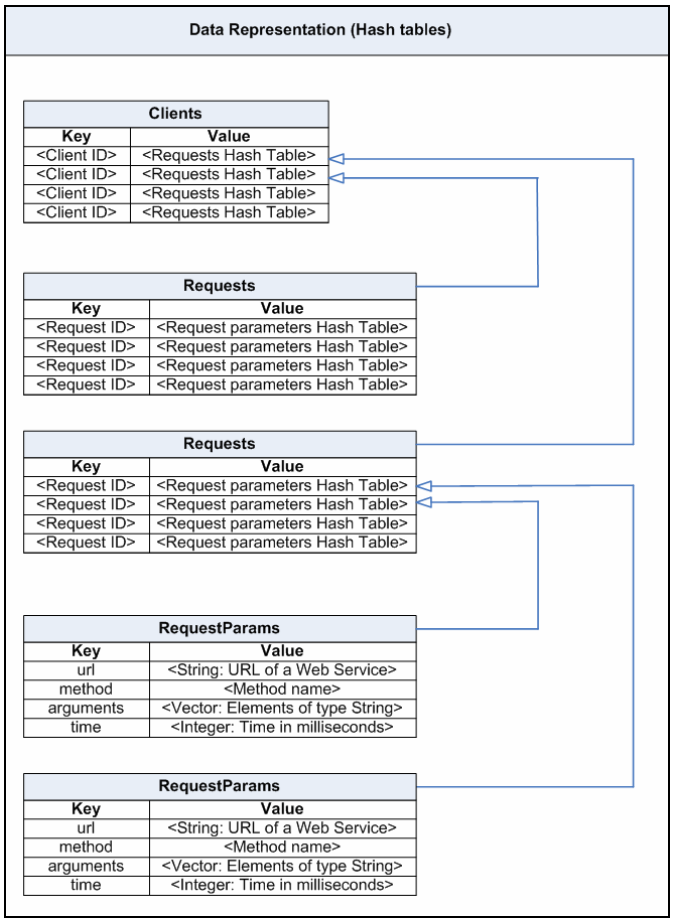

Figure 3. Clients' Requests Representation

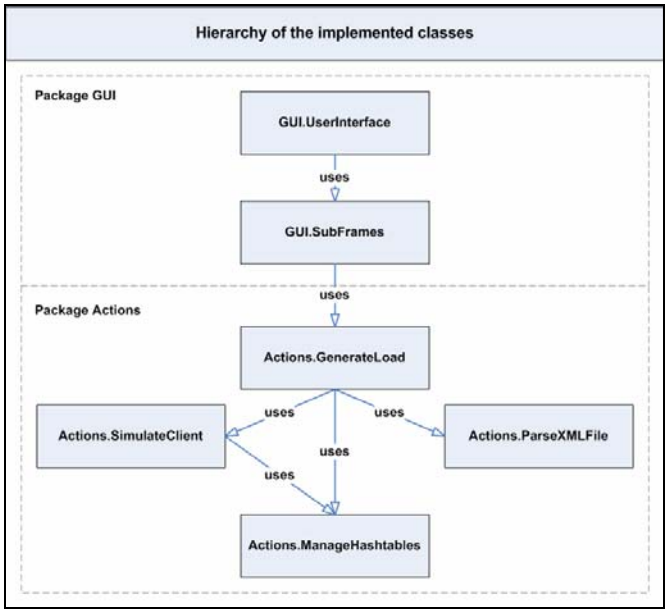

Figure 4. Hierarchy of the implemented classes

3.4. User Interface

The user-friendly interface of the Clients Load Generator & Simulator is developed by using the Java Swing Library. The windows' depth is two which allows the user to navigate easily. The main screen of the application suggests a choice of four options – to *create a single client's behavior*; to *create a multiple clients' behavior*; to *simulate clients' behavior*; and to *exit* the tool. Figure 5 shows the main window of the tool and Figure 6 represents the interface that creates a single client's behavior.

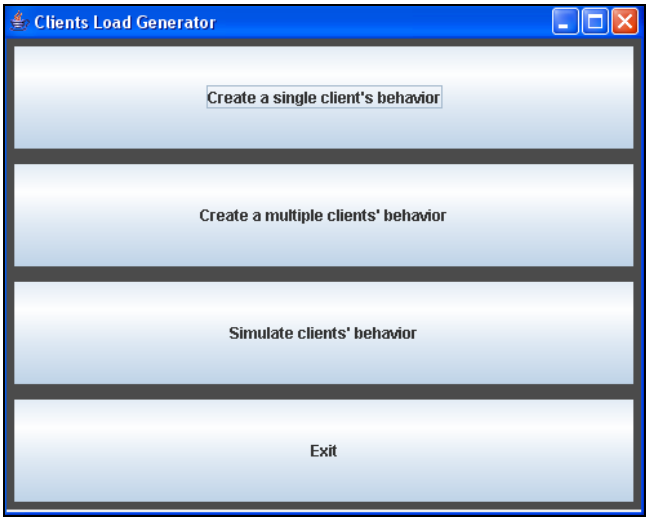

Figure 5. Main window of the Clients Load Generator and Simulator

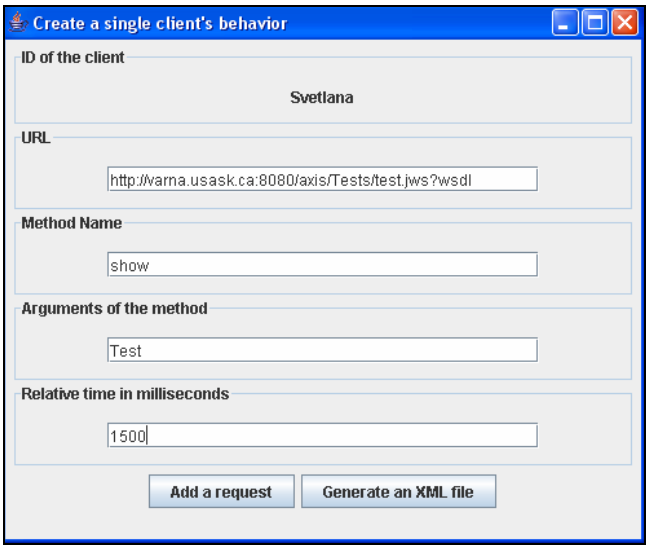

Figure 6. Window that creates a single client's behavior

The application checks if the user has entered the required values and if not, an appropriate message is displayed. After

each action, the user is able to see the result of the operation – whether it is successful or not.

The user is able to select RPC style or Document style of communication (Figure 7) when starting the simulation of the clients' behavior.

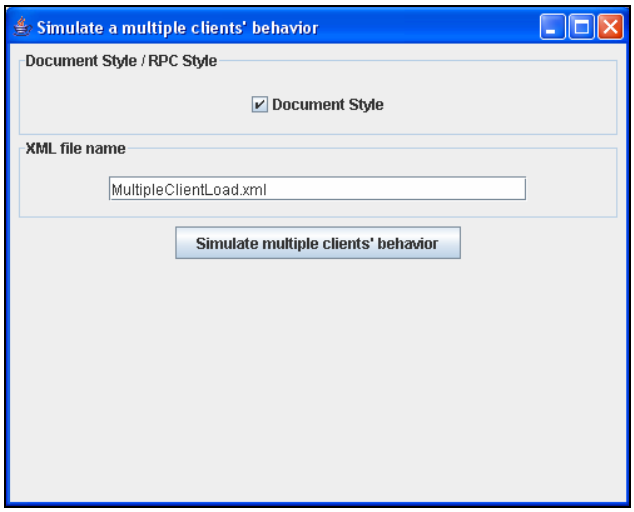

Figure 7. Window that starts the simulation of the clients

3.5. How to compile and how to run

- In order to compile the application, go to folder *Project/LoadGenerator/build/classes/* and run batch file *compile*;
- In order to execute the application, go to folder *Project/LoadGenerator/build/classes/* and run batch file *run*;
- The XML files are generated in folder *Project/LoadGenerator/build/classes/Load/*. The names of the created single client load files are *SingleClientLoad<client ID>.xml*. The name of the generated multiple clients' load file is *MultipleClientLoad.xml*;
- The obtained results of the multiple clients' load simulation can be found in folder *Project/LoadGenerator/build/classes/Results/* where each client has its own file with results. The name of the text file is *Results<client ID>.txt*.

4. EXPERIMENT

4.1. Assumptions

Two assumptions are made about the Web Services to be invoked by the Clients Load Generator and Simulator:

- To take as arguments String data types;
- The result of the invocation to be of String data type.

These requirements narrow the possible data types and at the same time allow the behavior of the services to be as real Web Services.

4.2. Experiment Setups

Eight experiment setups have been designed for each framework – RPC and Document, in order to test the Clients Load Generator & Simulator:

Type 1: Clients & Web Services are located in one machine:

- Scenario 1: 1 client – 1 request
- Scenario 2: 1 client – Many requests
- Scenario 3: Many clients – 1 request
- Scenario 4: Many clients – Many requests

Type 2: Clients & Web Services are distributed in different machines:

- Scenario 5: 1 client – 1 request
- Scenario 6: 1 client – Many requests
- Scenario 7: Many clients – 1 request
- Scenario 8: Many clients – Many requests

These experiments represent the full number of situations that could happen when the Clients Load Generator and Simulator is used. The evaluation of the tool gives an overview of the behavior of the Web Services in the different cases. The real-world situation is scenario 8 that represents many clients calling many distributed Web Services. However, it is interesting to observe the other cases as well since differences in the behavior of the services might occur.

The RPC style and the Document style are tested with exactly the same Web Services, client load, and load distribution. For RPC, the used framework is Apache Axis 1, and for Document style, the framework is Apache Axis 2.

5. RESULTS

The results of the client load simulation are saved automatically in text files. There is a separate result file for each client. The file contains the following values: client ID, request ID, Execution time, Start execution time, End execution time, Result of the execution, URL of the Web Service, Method name. If the Web Service is not reachable, the execution time is set to *-1* and the result is set to *null*. An example of a result file if shown on Figure 8.

The results of the conducted 16 experiments confirm that the Clients Load Generator & Simulator could be used for creating clients' behavior in XML format and for invoking Web Services. For Axis 1, when the clients and the services are located in the same machine, the response time is higher than the response time of the services when they are distributed. The reason for this is the increased overhead since the same machine has to send the clients' calls & to handle the requests. However, when the framework is Axis 2, there are no significant differences in the response times obtained with distributed and not distributed load. Consequently, the framework deploying the services is crucial as well.

The most important scenario is many distributed clients – many requests. It represents a real-case clients' behavior. However, the execution time varies significantly – from 15 milliseconds to 10-13 seconds for both styles (RPC and Document). The reason for this result could be the creation of the sockets, the garbage collection, and even the configuration of the Tomcat and the axis settings. A comparison between the obtained results is made in table 1 and table 2 as well as in Figures 9, 10, and 11.

ResultsSvetlana1 - WordPad

File Edit View Insert Format Help

Courier New 10 Western

No	ClientID	RequestID	Exec(ms)	Start time	End time	Result	URL	Method
1	Svetlana1	number_1	1157	Tue May 09 12:56:56 CST 2006	Tue May 09 12:56:57 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
2	Svetlana1	number_1	844	Tue May 09 12:57:01 CST 2006	Tue May 09 12:57:01 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
3	Svetlana1	number_1	1015	Tue May 09 12:57:05 CST 2006	Tue May 09 12:57:06 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
4	Svetlana1	number_1	985	Tue May 09 12:57:12 CST 2006	Tue May 09 12:57:13 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
5	Svetlana1	number_1	954	Tue May 09 12:57:17 CST 2006	Tue May 09 12:57:18 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
6	Svetlana1	number_1	1016	Tue May 09 12:57:21 CST 2006	Tue May 09 12:57:22 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
7	Svetlana1	number_1	1016	Tue May 09 12:57:25 CST 2006	Tue May 09 12:57:26 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
8	Svetlana1	number_1	1031	Tue May 09 12:57:29 CST 2006	Tue May 09 12:57:30 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
9	Svetlana1	number_1	1000	Tue May 09 12:57:32 CST 2006	Tue May 09 12:57:33 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
10	Svetlana1	number_1	953	Tue May 09 12:57:37 CST 2006	Tue May 09 12:57:38 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
11	Svetlana1	number_1	1047	Tue May 09 12:57:41 CST 2006	Tue May 09 12:57:42 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
12	Svetlana1	number_1	969	Tue May 09 12:57:46 CST 2006	Tue May 09 12:57:47 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
13	Svetlana1	number_1	1000	Tue May 09 12:57:50 CST 2006	Tue May 09 12:57:51 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
14	Svetlana1	number_1	1000	Tue May 09 12:57:54 CST 2006	Tue May 09 12:57:55 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
15	Svetlana1	number_1	1140	Tue May 09 12:57:59 CST 2006	Tue May 09 12:58:00 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
16	Svetlana1	number_1	984	Tue May 09 12:58:03 CST 2006	Tue May 09 12:58:04 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
17	Svetlana1	number_1	1015	Tue May 09 12:58:07 CST 2006	Tue May 09 12:58:08 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
18	Svetlana1	number_1	1031	Tue May 09 12:58:11 CST 2006	Tue May 09 12:58:12 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
19	Svetlana1	number_1	937	Tue May 09 12:58:18 CST 2006	Tue May 09 12:58:19 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show
20	Svetlana1	number_1	953	Tue May 09 12:58:23 CST 2006	Tue May 09 12:58:24 CST 2006	Hello test	http://varna.usask.ca:8080/axis/Tests/test.jws?wsdl	show

For Help, press F1

Figure 8. Example: Results of simulation

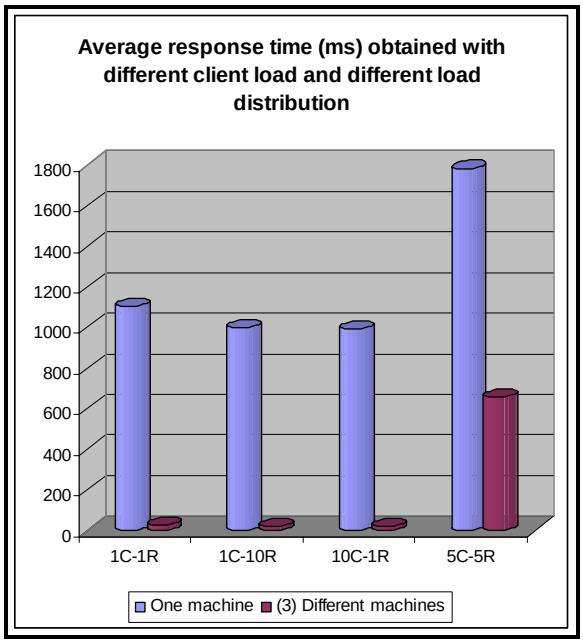

Figure 9. Average response time obtained during each experiment (Axis 1)

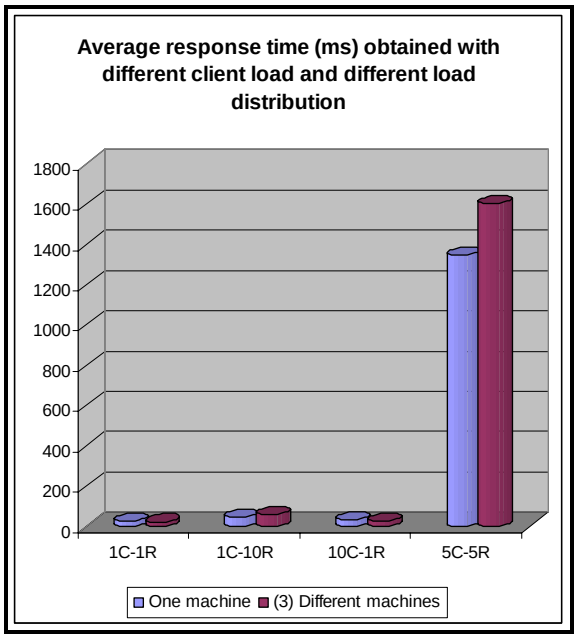

Figure 10. Average response time obtained during each experiment (Axis 2)

Experiment Type	Scenario	Min response time, ms	Average response time, ms	Max response time, ms	Number of repetitions
One machine	1 client – 1 request	875	1104.7	3250	20
	1 client – 10 requests	875	999.6	1203	20
	10 clients – 1 request	844	993	1157	20
	5 clients – 5 requests	969	1782.3	4094	10
(3) Different machines	1 client – 1 request	15	25	125	20
	1 client – 10 requests	0	21.1	94	20
	10 clients – 1 request	0	19.8	110	20
	5 clients – 5 requests	16	658.3	8875	10

Table 1. Experiment results (Axis 1)

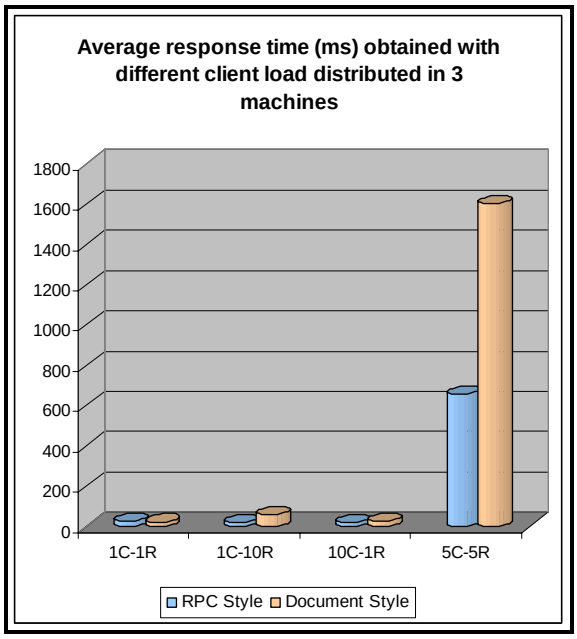

Figure 11. Comparison: RPC Style ←→ Document Style

Experiment Type	Scenario	Min response time, ms	Average response time, ms	Max response time, ms	Number of repetitions
One machine	1 client – 1 request	15	28.9	32	20
	1 client – 10 requests	15	48.11	266	20
	10 clients – 1 request	15	34.38	188	20
	5 clients – 5 requests	15	1348.65	6640	10
(3) Different machines	1 client – 1 request	15	24	32	20
	1 client – 10 requests	15	61.98	1125	20
	10 clients – 1 request	15	26.76	250	20
	5 clients – 5 requests	15	1603.36	12968	10

Table 2. Experiment results (Axis 2)

6. CONCLUSIONS AND FUTURE WORK

The presented Clients Load Generator and Simulator can be used as a tool during Web Services experiments. It operates as both XML clients' load creator and simulator. The application represents real-world clients' behavior since it considers the requests of one client as consecutive steps and the requests between different clients are simultaneous. The time between each call is controlled by a time tag within the XML clients' load representation. The Web Services are accessed through their WSDL description during the simulation period.

The tool is able to work on RPC and Document framework. The obtained results confirm that the proposed application can be used in experiments requiring invocation of Web Services. In addition, they represent some differences in the response times of the services when using Axis 1 and Axis 2.

Currently, the services that are invoked during the experiment period are developed in Java, based on Axis 1 (RPC style). Services, written in different languages such as C# and PHP could be added as well. In addition, the functionality of the tool might be extended to handle Web Services that follow the Axis 2 (Document) standard.