International Journal of Industrial Engineering and Management (IJIEM), Vol. 8 No 1, 2017, pp. 1-8
Available online at www.iim.ftn.uns.ac.rs/ijiem_journal.php
ISSN 2217-2661

UDK: 004.41

One Software Solution for Data Transfer Between Client and Server with Emphasis on Saving Memory and CPU Usage

Munir Šabanović
MSc., University of "Džemal Bijedić", Faculty of Information Technologies in Mostar
Sjeverni Logor br.12 88000 Mostar, BiH, munir.sabanovic@uninp.edu.rs
Muzafer Saračević
Ph.D., Department of Computer sciences, University of Novi Pazar
Dimitrija Tucovica bb, Novi Pazar, Serbia, muzafers@uninp.edu.rs
Emruš Azizović
Ph.D., Department of Computer sciences, University of Novi Pazar
University of Novi Pazar Dimitrija Tucovica bb, Novi Pazar, Serbia, emrus.azizovic@unhz.eu

Received (19.09.2016.); Revised (14.12.2016.); Accepted (01.03.2017.)

Abstract

The paper analyzes the memory usage and consumption of processor time when the user interacts with the application. We have analyzed the workload of the processor depending on the number of loaded data for different technologies of transfer (AMF, JSON or XML) from the server to the client. Apart from that, we have analyzed the occupation of memory depending on the number of loaded data and instantiated objects, the percentage of availability of memory that the class instances occupy. The change in the memory occupation is displayed graphically and numerically.

Key words: *client-server communication, data transfer, memory, CPU usage, XML.*

1. INTRODUCTION

This paper analyzes the workload of working memory with *Profile as* tool, and time engagement of the processor by using the *Task Manager* when authoring application that reads the data from the server *Wampserver2.4-x86* is started. *Profile as* tool enables memory availability assessment from multiple angles, one can see exactly how many instances of individual classes are active, the percentage of memory instances of a given class take up in relation to all living instance, whether the logic for removing references on inactive objects is activated immediately, gives average duration of individual methods. The wave form memory consumption is graphically presented where one can see the changes of memory dependency during the user's interaction with the application, as well as interaction of the application with the server. All of this can be traced for the three technologies of data transfer AMF, JSON and AMF. By carefully monitoring the process, we can see that the differences between transfer technologies do not exist when it comes to processor workload.

2. USED SCIENTIFIC METHODS AND PROCEDURES

In order to achieve the objectives and tasks of the research, the following scientific methods and procedures were used:

- By experimental application we have obtained results that demonstrate the effectiveness of new methods (in speed or saving memory space).
- Method of generalization is applied in the analysis of a number of cases where there is a general statement that applies to all cases. By the method specialization specific cases as specific examples were presented.
- Methods of deduction and induction are used in the course of the experimental research, where after the obtained results a conclusion about the new techniques and possibilities of their application was formed.

3. TECHNOLOGIES USED FOR MAKING AUTHORING APPLICATION

In order to create software solutions, *Flex* technology was used on the client side, PHP on the server side, while, for the data transfer were used three technologies AMF, JSON and XML. To analyze memory consumption is used *Profiler as*, while processor workload was monitored by the *Task Manager*. JSON and XML are used in many areas [6,7,11,12].
Flex is a highly productive, free open source software environment for creating executable version of expressive mobile, Web and computer applications. *Flex* enables one to create executable version of Web and mobile applications that share a common basic

code, thus shortening the time and cost of creating applications and long-term maintenance [1]. While *Flex* applications can be created using only the free *Flex SDK*, *Adobe Flash Builder* ™ can accelerate development with features such as intelligent code editing, gradual debugging, programs to optimize memory performance and visual design. *Flex* in its environment contains two languages .mxml for visualization and AS3 for functionality [1].

PHP server language is executed on the server side. It is named a scripting language because it is written in the form of scripts and was purpose built for use on the Web [7]. It can be written in separate files, or be inserted within HTML. The PHP processor on the Web server is interpreted by the PHP code, and Web server on the exit emits HTML or other types of data that the web browser on the client side can understand. A copy of the HTML page is sent directly from the server to the client computer, while the PHP code is not sent directly but is previously translated into a form that the client computer will know how to interpret. HTML parts in the script are left as they are until PHP code is not interpreted and executed with the PHP processor and the result of execution is sent to the client. PHP code has great possibilities, from communicating with other computers, creating images, databases access, work with the graphics, creating desktop applications using PHP-GTK extensions all the way to reading and writing files.

PHP does not have its owner; it is a free language, a group of enthusiasts gathered and made the PHP. For AMF technology, they have taken the Action Message Format and made serializers that correspond to action message format in PHP.

AMF or *Action Message Format* is a binary format that is used for serialization of objects and sending messages between the client and the remote service. Action Script 3 language has classes for encoding and decoding of the AMF format. *Adobe Systems* has released AMF binary protocol specification on 13 December 2007., and announced that it will support the developer community to make this protocol for all major server platforms. Thus, today there is AMF support for platforms written in Java, PHP, .NET and other languages. This paper will focus its attention on the PHP language because it is the chosen server platform.JSON or *JavaScript Object Notation* is a very simple text format for the exchange of data between server and client [6.10]. It is easy to parse and independent from any other programming language. Parsing of the JSON format can be performed using the built-in JavaScript functions [11,12]. Compared to XML format, it is faster, shorter and there are no reserved words [8.9]. XML, or *Extensible Markup Language*, is a language for creating electronic documents. One XML document is a hierarchy of XML elements. Each element represents part of the information contained in the document [2]. The content of the XML document includes: processing instructions, elements, attributes and comments.

Adobe Flex profiler can identify bottlenecks and memory leaks in your application. The user interacts with the application, the *profiler* records the state of the application including the number and size of facilities, the number of called for methods, archives the time methods spend from the moment when they were called for to the moment when they are not active. Using the *profiler*, one can see how much time a method is active, or if the same method was called for multiple times, one can get the average time for the methods by profiling section. If it is discovered that the method causes a performance bottleneck, then the user can try to optimize the method. This tool enables following the order of calling for methods and to determine which method is unnecessarily called for. Also, the profiler provides information to the user on how many active instances there are as well as the total number for each class individually. To avoid multiple objects of a class to instantiate multiple times simultaneously, if they are not required, then the class is defined as a singleton class. If an object takes up a lot of memory, then there is a possibility of optimizing memory consumption.

In comparison with the neighboring states of an application, which is achieved by current recordings, one can discover that some objects, which are allocated amount of the memory, are not needed to the application. To free the memory and thereby eliminate memory leaks there is a logic which removes any residual references to a given object. So, when it is determined that an object which is placed in memory, is no longer needed, then there is logic that is destructing the given object. This logic is called *garbage collection* or *garbage collector*.

It is recommended that the profiling is done during the development of the application in order to determine the bottleneck in the application which influences the performance and in order to optimize the code as soon as possible.

The application is implemented in the object model. The central position in the object programming occupies the class or class instance - the object. Classes are defined as building blocks and interconnections of models being described. Among the relevant experts from object-oriented programming there is no agreement when it comes to regarding the object and therefore generally is resorted to an empirical definition of the object. In the opinion of relevant authors (G. Booch), each object is defined using the identity, state and behavior. Object identity is something that makes the objects of the same or different classes differs from one another. Identity is unique so that two objects in the same memory space cannot have the same identity even though they may be located in the same state and to have the same behavior. The state of the object is described using specific values of the class member's data and is result of the operations that were carried out on the object in the past and determines the behavior of the object in the present and the future. For example, in the stack a reading operation cannot be executed if there are no elements in the stack but the operation writing in can be done.

The object behavior is described using operations and the implementation of the operation is called the

"method". Class can have one operation, several or no operation at all. For example, in the working library with windows, as is the case with Java's awt package, all *Rectangle* class objects can be moved, can be scaled or their features can be tested. Usually (but not always), the call for operation over the object changes its data or state. The behavior of the object, in response to the activation of some operations (initiative, message) depends on three factors:

- the very initiative
- state of the given object
- state of other objects from the same or different classes which are not objects-members, but have an impact in the initiative framework.

So it can consult a method or state of a third object.
It is also necessary to note that an object can be active and passive. If it has installed behavior then it refers to the active object as opposed to a set that is passive because there are no methods that are built-in within the style. Thus the behavior of the object is determined by the operations, where specific behavioral pattern depends both on the operation and of the current state. Activating the operation is performed with the so called message. Class features i.e., object can be descriptive and procedural. The *Stack* class has the descriptive characteristics of a number of elements of the stack as well as the elements. Procedural characteristics show what the stack can do: read, enter, delete, etc.

4. THE APPLICATION DESIGN AND FUNCTIONALITY

Figure 1. shows the application working environment. Application working environment contains a grid with columns *Id*, First name, Last name, Department, Index no. and Date, Figure 2. The columns are loaded data from the database, which is located on the *Wampserver 2.4-x86* server. Number of loaded data is selected from the *ComboBox* within the range 100-76000 data.

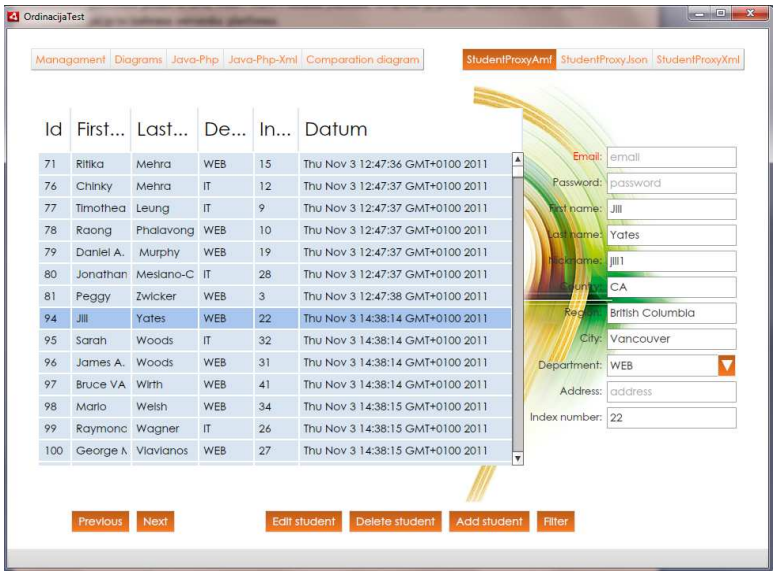

Figure 1. Application working environment

Id	First Name	Last Name	Department	Index no.	Datum
71	Ritika	Mehra	WEB	15	Thu Nov 3 12:47:36 GMT+0100 2011
76	Chinky	Mehra	IT	12	Thu Nov 3 12:47:37 GMT+0100 2011
77	Timothea	Leung	IT	9	Thu Nov 3 12:47:37 GMT+0100 2011
78	Raong	Phalavong	WEB	10	Thu Nov 3 12:47:37 GMT+0100 2011
79	Daniel A.	Murphy	WEB	19	Thu Nov 3 12:47:37 GMT+0100 2011
80	Jonathan	Meslano-Crookston	IT	28	Thu Nov 3 12:47:37 GMT+0100 2011
81	Peggy	Zwicker	WEB	3	Thu Nov 3 12:47:38 GMT+0100 2011
94	Jill	Yates	WEB	22	Thu Nov 3 14:38:14 GMT+0100 2011

Figure 2. Component *DataGrid* on the application working environment

The application contains two Bar buttons. The first stateSelector Bar button contains five states: Management, Diagrams, Java-PHP, Java-PHP-XML and Comparation diagram.

This control is defined by code block[3]:

```
<s:ButtonBar
top="25" left="30"
dataProvider="{statesProvider}"
```

```
labelField="label"
id="stateSelector"
change="stateSelector_changeHandler(event)"
/>
```

DataProvider data supplier is a series ofstatesProvider which is defined by the following code block:

```
<s:ArrayList id="statesProvider">
<fx:Object label="Managament" state="managament"/>
<fx:Object label="Diagrams" state="diagrams"/>
<fx:Object label="Java-Php" state="javaPhp"/>
<fx:Object label="Java-Php-Xml" state="javaPhpXml"/>
<fx:Object label="Comparation diagram"
state="comparation"/>
</s:ArrayList>
```

Authoring application loads the data from two different servers, Wampserver2.4-x86, which works with PHP-player and Niti server which works with Java. If the state Management is active then the application working environment is displayed. However, if the second state Diagrams is active then graphs for the three technologies of data transmission are displayed, AMF, JSON and XML. The state JavaPHP shows JSON charts when there is communication of the client authoring application with the server Wampserver2.4-x86 and Niti server.

If the fourth state JavaPhpXml is active, then graphs are displayed which measure the time of n data, when the data on the part of the server is formatted in XML. If the Comparation diagram button is active, then a comparison chart of AMF-JSON-XML query execution is displayed.

The second Bar button selectProxyBar, changes its appearance depending on the application operating conditions. The Management state displays three buttons StudentProxyAmf, StudentProxyJson and StudentProxyXml. These three buttons offer the choice of data transfer technology from the database into the application in AMF, JSON or XML.

Button controls, whose labels are Previous, Next, Edit student, Delete student, Add student and Filter, respectively enable display of previous data block from the database in the grid, the next data block, editing the selected item in the grid and the base, deleting rows in the database, adding new-student in the database and filtering data in the grid. Depending on the state normal or search in the class instances StudentControlComponent and StudentManagerComponent.

The filter control can take two values for label Filter or Back. TextInput fields and ComboBox serve to display values from the selected rows of the grid, changes to the content in the selected row and adding new content. In the state normal are displayed all the TextInput controls and ComboBox, while the state search does not display all TextInput fields.

5. SYSTEM RESOURCES ANALYSIS: PROCESSOR AND MEMORY

The application analyzes the number of active instances, the percentage of availability of memory instances of different classes that exist in the program, instances that take up a lot of memory, the average period the methods occupy during all sections of engagement, period in sections.

Based on these parameters, one can optimize the code. The chart can trace changes in memory consumption on the basis of the activities of each component in the application.

This is achieved by using the Profile as tool which allows tracking of each application segment, provides data on the number of instantiated objects, memory usage, identifies performance within the application.

Field for running Profile as tool, of the Adobe Flesh Builder located in the Run menu, Figure 3.

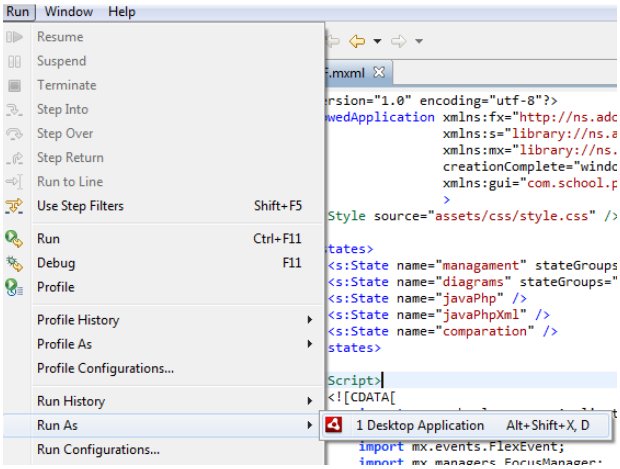

Figure 3. Starting the application analyzer from the *Run* menu.

When you run the application, the StudentAMF.mxml class object is instantiated. The sudden jump of memory occurred in the graph at the moment of loading 500 objects from the server to the client application. The whole application is not larger than 20 MB, which can be seen in Figure 4.

Figure 4. Shows the maximum memory usage, red line, current memory usage, the blue line, and change of the memory consumption over time, showing a memory space occupied by the applications, 20 480 Kb.

Current memory usage is 13860Kb, as shown in the field Current Memory: 13860Kb. Maximum occupied memory for application was 13860Kb, the red line in the graph.

Class StudentVO has 500 active instances (Figure 5.), and occupies 67.7% instances of the total number of instantiated objects in memory of different classes.

Class	Package (Filtered)	Cumulative Instances	Instances	Cumulative Memory	Memory
StudentVO	com.school.vo	1000 (23.33%)	1000 (57.97%)	76000 (25.32%)	76000 (39.55%)
Vector.<*>	__AS3__.vec	1244 (29.02%)	226 (13.1%)	59384 (19.79%)	18176 (9.46%)
MethodClosure	builtin.as$0	1253 (29.23%)	0 (0.0%)	50144 (16.71%)	0 (0.0%)
StudentTextInputSkin	com.school.puremvc.view.gui.skins	11 (0.26%)	11 (0.64%)	14740 (4.91%)	14740 (7.67%)
ButtonBarButtonSkin	com.school.puremvc.view.gui.skins	8 (0.19%)	8 (0.46%)	11200 (3.73%)	11200 (5.83%)
StudentButtonSkin	com.school.puremvc.view.gui.skins	7 (0.16%)	7 (0.41%)	9828 (3.27%)	9828 (5.11%)
StudentDataGridSkinInnerClass5	com.school.puremvc.view.gui.skins.grids	6 (0.14%)	6 (0.35%)	8376 (2.79%)	8376 (4.36%)
StudentDataGridSkinInnerClass9	com.school.puremvc.view.gui.skins.grids	16 (0.37%)	16 (0.93%)	6464 (2.15%)	6464 (3.36%)
HLayoutElementFlexChildInfo	HorizontalLayout.as$125	67 (1.56%)	0 (0.0%)	6432 (2.14%)	0 (0.0%)
ButtonBarSkinInnerClass1	com.school.puremvc.view.gui.skins	4 (0.09%)	4 (0.23%)	5856 (1.95%)	5856 (3.05%)
Property	flashx.textLayout.property	88 (2.05%)	88 (5.1%)	3872 (1.29%)	3872 (2.02%)
Vector.<int>	__AS3__.vec	46 (1.07%)	18 (1.04%)	3384 (1.13%)	1624 (0.85%)
ButtonBarSkinInnerClass2	com.school.puremvc.view.gui.skins	2 (0.05%)	2 (0.12%)	2928 (0.98%)	2928 (1.52%)
ButtonBarSkinInnerClass0	com.school.puremvc.view.gui.skins	2 (0.05%)	2 (0.12%)	2928 (0.98%)	2928 (1.52%)
ButtonBarSkin	com.school.puremvc.view.gui.skins	2 (0.05%)	2 (0.12%)	2640 (0.88%)	2640 (1.37%)
PriorityBin	PriorityQueue.as$518	88 (2.05%)	88 (5.1%)	2112 (0.7%)	2112 (1.1%)
StudentAMF		1 (0.02%)	1 (0.06%)	2104 (0.7%)	2104 (1.09%)
StudentDataGridSkinInnerClass4	com.school.puremvc.view.gui.skins.grids	5 (0.12%)	5 (0.29%)	2020 (0.67%)	2020 (1.05%)
StudentDataGridSkinInnerClass2	com.school.puremvc.view.gui.skins.grids	5 (0.12%)	5 (0.29%)	2020 (0.67%)	2020 (1.05%)
LayoutElementFlexChildInfo	VerticalLayout.as$128	21 (0.49%)	0 (0.0%)	2016 (0.67%)	0 (0.0%)
StudentManagerComponent	com.school.puremvc.view.gui	1 (0.02%)	1 (0.06%)	1484 (0.49%)	1484 (0.77%)
StudentDataGridSkin	com.school.puremvc.view.gui.skins.grids	1 (0.02%)	1 (0.06%)	1468 (0.49%)	1468 (0.76%)
StudentComboSkin	com.school.puremvc.view.gui.skins	2 (0.05%)	1 (0.06%)	1428 (0.48%)	1388 (0.72%)
BackgroundSkin	com.school.puremvc.view.gui.skins	2 (0.05%)	1 (0.06%)	1364 (0.45%)	1324 (0.69%)
Vector.<Number>	__AS3__.vec	12 (0.28%)	8 (0.46%)	1360 (0.45%)	992 (0.52%)
StudentTableComponent	com.school.puremvc.view.gui	1 (0.02%)	1 (0.06%)	1324 (0.44%)	1324 (0.69%)
StudentControlComponent	com.school.puremvc.view.gui	1 (0.02%)	1 (0.06%)	1324 (0.44%)	1324 (0.69%)
EnumPropertyHandler	flashx.textLayout.property	50 (1.17%)	50 (2.9%)	1000 (0.33%)	1000 (0.52%)
SizesAndLimit	HorizontalLayout.as$125	31 (0.72%)	0 (0.0%)	992 (0.33%)	0 (0.0%)

Figure 5. Shows a list of all the objects that currently exist in the application.

In Figure 5., there is a list of classes in application which have instantiated objects in memory, a package which includes classes, total number of instantiated objects for each class, the number of active instances of each class, size of the memory occupied by objects of each class expressed in bytes and in percentages with respect to all instances that have existed, current memory usage occupied by active objects of a given class. This table shows that most memory occupies class instance StudentVO, which amount 500 stored in the working memory. Graphic components take up a lot of memory, because the graphics classes are bulky and therefore it is necessary to reduce the number of these classes to a minimum so that the total memory usage would be lesser and when drawing they take up a lot of processor time.

Pressing the Next button in the application, Figure 1., 500 new instances are called for from the database. Now the table shows that the total loaded amount is 1000 objects and that there are1000 active instances.

However, the Profile as logic has prepared 500 instances for destruction, it is waiting for a moment when they would be destroyed, however, if one presses the GabrageCollector button, then manually is run the logic to remove all references on objects that are no longer used and thus eliminate memory leaks.

So, when one loads 500 new instances from the database, memory has increased to 9710Kb and then dropped to 7079kb because the GarbageCollector cleaned 500 instances from the memory which are no longer used. In the table, objects of class StudentVO occupy 57.97% of the memory, skins for text fields of the class StudentTextInputSkin occupy 7.67% of the memory. However, the proxy StudentProxyJson occupies 0.05% of the memory. These are classes that do not have heavy objects and therefore take up little memory. There is one instance of the controller StudentControlComponent which occupies 0.69% of the memory. For ButtonBarSkin button in total seven instance are instantiated, in the memory currently exist 7 occupying 1.37% of the memory.

When one would calculate the memory of all skins, StudentComboSkin 0.91%, ButtonBarSkinInnerClass1 1.93%, etc., it can be seen that they occupy a lot of memory.

By clicking the Diagrams button in ButtonBar, Figure 6., there is a jump in memory because another component is included. If the Java-PHP button is clicked there is a graphics rebound, which displays memory usage; the case is the same when involving other components.

.

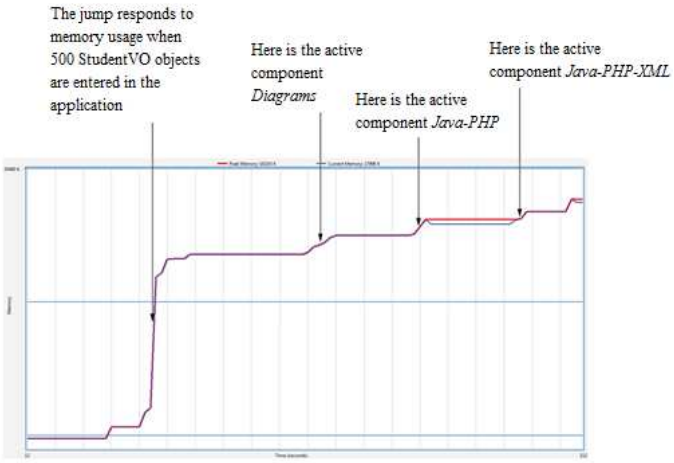

Figure 6. Shows memory space occupied by the application when the user interacts.

Flex does not reduce memory when Diagrams or other component has a value null, because the virtual machine evaluates how much memory it consumes and holds reserved memory. The Garbage collector did not finish the deletion job because it estimated that memory will be needed.

When it comes to processor time, the largest amount of processing time is spent when loading data from the server to the application. In free mode, processor usage is small, image 7a.

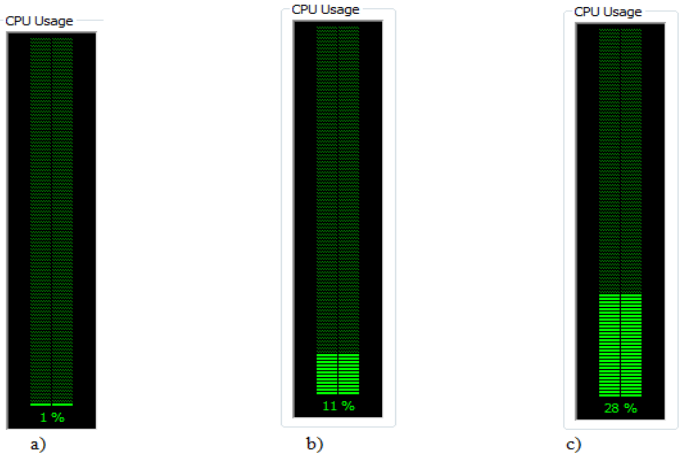

Figure 7. CPU usage a) in free mode, b) when 500 objects are loaded, class *StudentVO*, c) when creating a chart, for any technology transfer AMF, JSON or XML, when the component *Diagrams* is active.

When the Next button is pressed on the working application and at the same time are loaded new 500 objects of the StudentVO class, then the processor is considerably burdened serving the application, as shown in Figure 7b. Comparing processor employment in Figures 7a and 7b shows that the load in the second case is 10% higher.

If the component Diagrams is active, at the same time data loading period into the application is measured provided that the iteration is repeated 50 times, then

the processor time is considerably spent, as can be seen in Figure 7c.
Testing is performed on computer with the following performances: CPU - QuadCore AMD Phenom X4 9550, 2200 MHz (11 x 200), L1 64 KB per core, 512KB L2 per core (On-Die, ECC, Full-Speed), 2 MB L3 (On-Die, ECC, NB-Speed), RAMMemory - 3 Gb, Graphic card - nVIDIAGeForce 9400 GT. The environment of authoring application was used for the test.

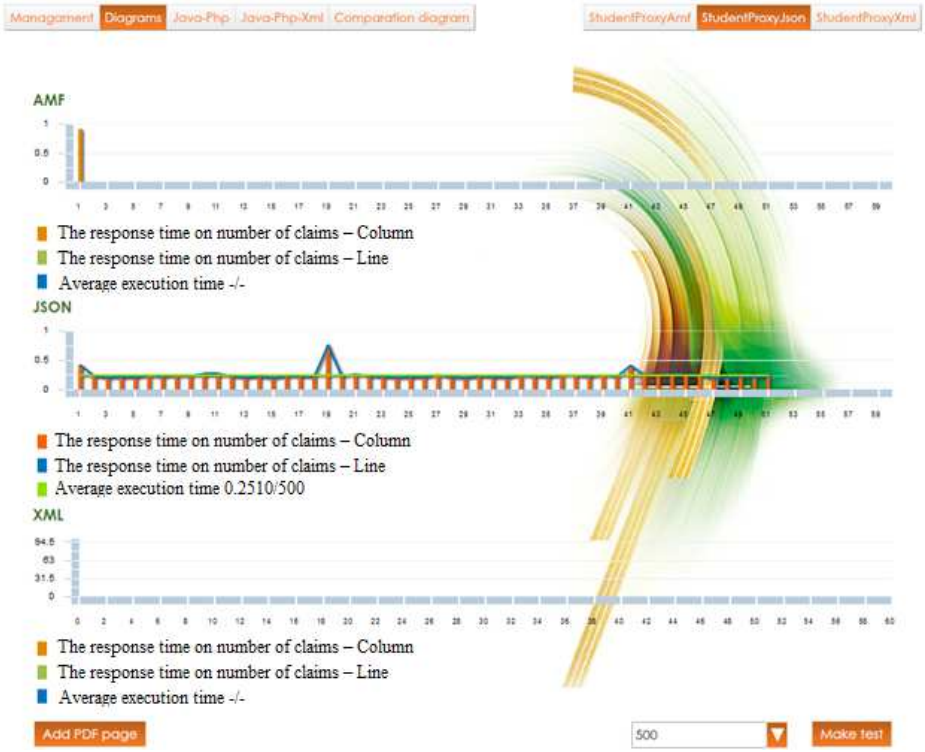

Figure 8. The load data from the server in JSON format, and at the same time processor workload is measured.

6. CONCLUSION

The paper analyzes the memory usage and consumption of processor time when the user interacts with the application.
For the analysis of the working memory load the *Profile as* tool was used which gives wide scope for reviewing different aspects of the working memory workload, gives a clear picture to the author of the application how to optimize the code and thus increase the operating speed of the application and prevent memory leaks.
The conclusion is that there is a rapid increase in memory consumption when loading data from the server. Depending on the amount of working memory on the client's computer one needs to load the optimal number of data from the database.
During the preparation of the application it is recommended to use *Profile as* tool in order to optimize the time and prevent memory leak. In the application it is determined that graphic components take up a lot of memory and processing time, it is highly recommended to minimize the skin class. When measuring the mean time, it is necessary to select the optimal number of iterations so the processor would not be burdened too much.

7. REFERENCES

[1] *Using ADOBE FLEX 4.6. Tutorial from official web site help.adobe.com/ (visited 19.11.2014.).*
[2] *Zimmermann O., Tomlinson M., Peuser S.. Perspectives on web services: applying SOAP, WSDL and UDDI to real-world, Springer, 2003.*
[3] *Hall C., ActionScript Developer's Guide toPureMVC, O'Reilly Media; 1 edition, 2011.*
[4] *C++ FAQ. Kept by Marshall Cline, retrieved from www.parashift.com/c++-faq-lite/ (visited 19.11.2014.)*
[5] *Encryption Basics. EFF Surveillance Self-Defense Project, Surveillance Self-Defense Project, n.d. Web. 06 Nov. 2013., ssd.eff.org/tech/encryption (visited 19.11.2014.)*
[6] *Florescu D, Fourny G. JSONiq: The History of a Query Language, IEEE internet computing, 17 (5), 2013, pp. 86-90.*
[7] *Pettit, JB, Marioni, JC. BioWeb3D: an online webGL 3D data visualisation tool, BMC bioinformatics, 14 (1), 2013, pp. 185.*
[8] *Jorstad, I, Bakken, E, Johansen, TA, Performance evaluation of JSON and XML for data exchange in mobile services, WINSYS 2008: proceedings of the international conference on wireless information networks and systems, 2008, pp. 237-240.*
[9] *Rodrigues, C, Afonso, J, Tome, P, Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML, enterprise information systems, Communications in Computer and Information Science, 220 (1), 2011, pp. 162-169.*
[10] *Adamanskiy A, Denisov A. EJDB - Embedded JSON database engine, 2013 fourth world congress on software engineering (WCSE), 2013, pp. 161-164.,*
[11] *Merelo-Guervos J, Castillo J, Laredo A. et al.. Asynchronous Distributed Genetic Algorithms with Javascript and JSON, Conference: IEEE Congress on Evolutionary Computation Location: Hong Kong, 2008, pp. 1372-1379.*

[12] *Jun Y, Zhishu L, Yanyan M. JSON Based Decentralized SSO Security Architecture in E-Commerce, International Symposium on Electronic Commerce and Security Location: Guangzhou, Proceedings of the international symposium on electronic commerce and security, 2008, pp. 471-475.*

[13] *http://www.tricedesigns.com/2011/11/07/AMF-vs-JSON-in-air-mobile-applications/ (visited 25.11.2014)*

[14] *http://www.jamesward.com/2007/04/30/ajax-and-flex-data-loading-benchmarks/ (visited 25.11.2014.)*

Softversko rešenje za prenos podataka između klijenta i servera sa fokusom na racionalno korišćenje memorije i procesora

Munir Šabanović, Muzafer Saračević, Emruš Azizović

Primljen (19.09.2016.); Recenziran (14.12.2016.); Prihvaćen (01.03.2017.)

Apstrakt

U radu je analizirana upotrebu memorije i vremensko angažovanje procesora pri interakciji korisnika sa aplikacijom. Predmet analize predstavlja opterećenje procesora u zavisnosti od broja opterećenih podataka za različite tehnologije prenosa (AMF, JSON ili KSML) od servera ka klijentu. Pored toga, analizirana je zauzetost memorije u zavisnosti od broja obima podataka i instanciranih objekata, procenat dostupnosti memorije u zavisnosti od klase instance koju zauzima. Promene zauzetosti memorije prikazana je kako grafički tako i numerički.

Ključne reči: *klijent-server komunikacije, prenos podataka, memorija, korišćenje procesora, XML*