The Effect of Applying Software Design Patterns on Real Time Software Efficiency

Muhammad Ehsan Rana
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, 43400
Serdang, Malaysia
ranaehsan@gmail.com

Wan Nurhayati Wan Ab. Rahman
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, 43400
Serdang, Malaysia
wnurhayati@upm.edu.my

***Abstract*—Real time applications are one of the fastest growing applications in the market due to their popularity, business value and the fact that web is their native environment. As a result, enhancing the performance of these applications has always been a concern for the IT industry. In this research, we took a closer look on the effect of design patterns on the performance of these applications using simulations as a research method. Two of the design patterns used by the researchers, namely, the Observer and the State design patterns, proved to be more effective in terms of software efficiency.**

Keywords—Design patterns; real time software; real time applications; software performance; software efficiency

I. Introduction

Web-dependent applications are among the most popular software applications in today's IT industry. The available web applications in the market outnumber the stand-alone applications with a huge margin. Among various different types of web-applications in the market, a category that stands out in terms of popularity and usage is social media. These internet based applications are growing faster than most of the other applications, mainly due to their high number of users and the strong linkage between the users of these applications [1].

Real time programming is the core of these applications (involving chatting, live commenting, gaming, photo and video sharing, etc.) because of their high dependence on interactivity [2]. The rise of the usage of real time programming took the attention of many researchers to the best practices of developing these types of applications. Hardware optimization, cloud technologies and practices such as multi-processing were some of the main areas of research for many companies. But there are many limitations in applying these types of practices, especially when it comes to hardware optimization, where cost is the biggest issue.

The next solution for real-time programming that might affect the way that they are being used is by considering the programming practices, where it requires more skilled programmers rather than optimized hardware, this will benefit the companies since it will save them from the limitations mentioned above. Using the right programming practices can affect the performance of the real-time applications, both by boosting the speed on the software level, and by using the maximum capabilities of the underlying hardware. There are various design pattern based solutions that can be used to improve the performance of such applications, but possibly one of the most well-known ones are the software design patterns known as GoF (Gang of Four) patterns.

Fundamental Research Grant (FRGS) by Ministry of Higher Education Malaysia under Project Title: "Architectural Patterns and Performance Anti-Patterns for a Systematic Migration Framework for Moving Legacy Applications to Cloud".

The undergoing research is an attempt to test the effect of three of these GoF design patterns (viz. State, Strategy and Observer) on the performance of real-time applications in terms of software efficiency. This paper is organized as follows: Section 2 analyzed the literature reviews, Section 3 described the research method, Section 4 discussed the findings and Section 5 concluded the paper.

II. Literature Review

A. *Real Time Programming*

"Real time is a level of computer awareness that a user senses as adequately immediate or that enables the computer to keep up with some external process (for example, to present visualizations of the weather as it constantly changes)." [3] Real-time applications are nearly used in most of the major software applications, in particular embedded systems, where a specific behavior is required on a timely manner. They can also be seen in GPS applications, chat rooms, mobile systems and etc.

Web-based applications are similar to other real-time applications, with one major difference that the response time is critical here. As a result, there have been lots of attempts to improve the performance and responsiveness of these applications using different methods. Hardware optimization is a primary response in these situations, but as they are not entirely cost-effective, software optimization is the next step. Software optimization can be done in different levels, starting with refactoring the code using programming best practices, to minimize the heavy processes by using certain policies. Software design patterns are among these best practices that are developed to enhance both the readability and the performance of the applications. Very few well documented simulation or researches on the effect of design patterns on software performance are available, but they should logically boost the performance due to their short syntax and their effective way of handling requests.

Real time applications can be used in any industry or type of programming, as long as the time factor is given due importance and no delay is expected in responses [4]. However there are software and hardware considerations when it comes to implementing these applications, since they have to deal with a high volume of data and sometimes the sensitive nature of the data they have to handle. There always has been an argument over choosing the right programming language for developing these applications. The argument has been on both the performance of these sorts of applications and the level of flexibility required programming these applications. Generally speaking, they can be developed using most of modern programming languages (although Java, C++, and Scala are the popular choices for this type of programming), but as it's mentioned before, picking up the right programming language actually depends on the purpose and scope of the application. For developing more solid and stable versions, static languages are preferred, while in some cases due to the nature of the application, dynamic programming languages are more suitable.

B. *Issues Associated with Real Time Applications*

Developing real time applications can be challenging since it deals with the real world entities, and these can cause real challenges to the programmers who need to develop the code in such a way that it can handle all different scenarios appropriately [5]. Another source of complexity comes from the fact that real-time applications are generally handling massive traffic that has to be handled efficiently to keep the program up and running, and responds to the request on a timely fashion. Below is a brief discussion on some of the main issues in developing real-time applications:

1) *Software Architecture Suitability*

It is important to develop the system in a way which is capable to take advantage of queuing systems and minimizing the number of nodes. These strategies help the system to maximize the performance and recover faster in case of any failure.

2) *Synchronous vs. Asynchronous*

Developing real-time applications using synchronous methods can cause application performance issues, as any new request will hold an open session until the response is provided, that will put the system on a heavy pressure. Asynchronous calls, on the other hand, will not require any open session, so they can be a better alternative for real-time programming.

3) *Compatible Hardware Resources*

As it's been discussed before, most of the real-time applications are handling heavy-loads of requests, so they require a powerful hardware infrastructure to handle this load. It is also important to use the elastic technologies, like cloud, which enable the companies to scale and expand as needed.

4) *Choosing the Right Operating System*

Choosing the right operating system is as important as choosing the right hardware, as the hardware capabilities should be supported by the operating system to allow the application for taking advantage of the entire hardware capabilities.

5) *Failure Recovery*

It often happens that real-time applications crash due to the load of traffic, therefore it is important to have a proper recovery plan (both on the software and hardware level) to minimize the damage to the user-experience.

C. *Software Design Patterns*

According to [6], "In software development, a pattern (or design pattern) is a written document that describes a general solution to a design problem that recurs repeatedly in many projects." Design patterns are well-known and well-defined solutions to some of the common design problems. Using these solutions makes it a lot easier for other programmers to read the code and understand the logic behind it, on the other hand they also help in improving the usability and easing the maintenance of the code [7].

In 1991, Gamma and Helm together with Ralph Johnson and John Vlissides published their famous book of Gang of Four (GoF), which until now (with some modifications) is considered as one the main reference books for design patterns. [8] Software design patterns are categorized under three main categories, creational, behavioral and structural [9]. Most of the performances issues in real-time programming are directly related to the behavior of the objects and not particularly with the application architecture or structure, researchers have narrowed down their focus on behavioral design patterns that affect the performance of the applications comparatively more significantly. For the sake of this research, researchers have selected three design patterns under the category of behavioral design patterns to test the hypothesis. The three selected design patterns are State, Strategy and Observer. Following is a brief description of these three design patterns.

State design pattern allows an object to alter its behavior when it's internal state changes. The object will appear to change its class [9] (Fig. 1).

Strategy design patterns allows applications to implement multiple algorithms and use them interchangeably, this gives the developers the ease and flexibility to choose their preferred method [9] (Fig. 2).

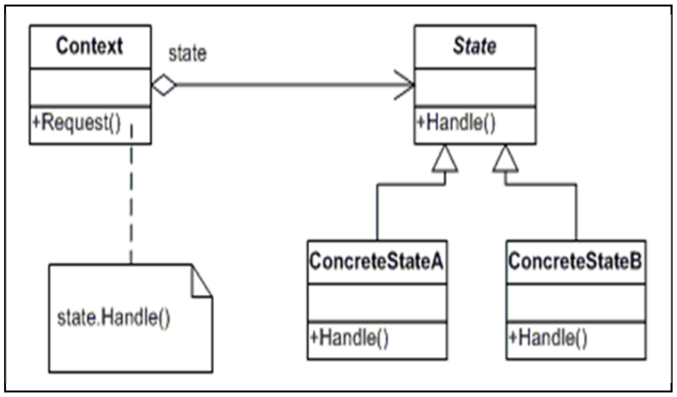


Fig. 1. State design pattern implementation.

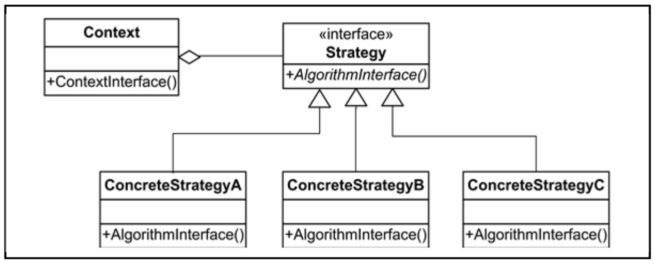


Fig. 2. Strategy design pattern implementation.

The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The Observer pattern assumes that the object containing the data is separate from the objects that display the data, and that these display objects observed changes in that data [9] (Fig. 3).

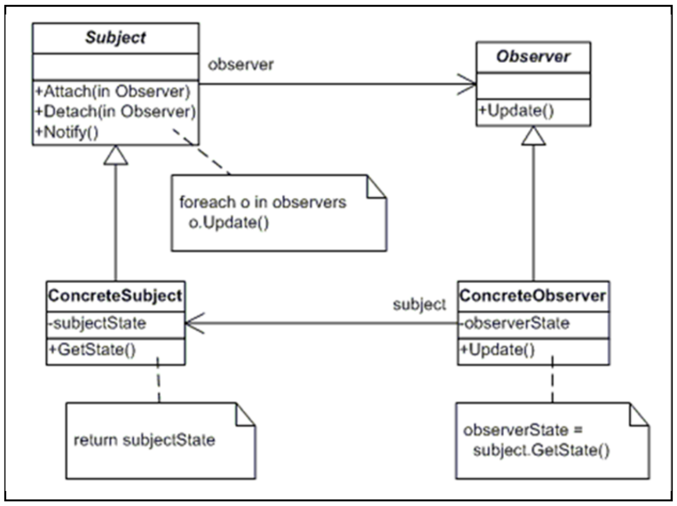


Fig. 3. Observer design pattern implementation.

D. *Measuring Software Performance*

Real time programs are dissimilar from other applications by the fact that the performance and as a result the response time of many of these applications are mission-critical where failure in responding can cause catastrophic damage. It is important to mention, that responding in a timely fashion doesn't necessary means fast, but it means delivering on the promised time period [10].

E. *Performance Testing*

When it comes to measuring the performance of applications, there are different performance testing techniques available that relies on various factors such as speed, scalability, stability, memory consumption, etc. Although depending on the requirements, other types of performance testing are also conducted in which we target the availability, response time and flexibility of applications, however what is important in performance testing is to find out the true nature of the application and its expected results, so that the right set of performance tests can be designed and performed.

F. *Real-Time Systems' Performance Testing*

As it's been described before, when it comes to real-time applications, the response time (getting the job done in a promised time period) is the most important factor; therefore most of the performance testing should be done with this idea in mind.

It is important to mention that response time relies on various variables, memory consumption is possibly one of them as every server (the machine that hosts the application) has a limited memory, and using this memory efficiently will increase the performance tremendously. Another important thing to state is that the performance of any application is the aggregate result of the performance of all functions in that application, and it is not relying only on a single component or a portion of the software.

G. *Difficulties and Challenges*

The overall performance of any application is the result of various different factors that include software, hardware, coding style, etc. Therefore measuring the performance of the application accurately is quite challenging.

Another difficulty in measuring the performance of an application, especially for a real-time application, is to create enough data load for this application, so that the results of testing are tangible.

III. Research Method

To evaluate the performance of the application, the designed simulation is implemented. Each test is performed twice, first by using the stated design pattern and secondly by running the code that simulates the same behavior without using the design pattern. The response time is measured using Google Chrome's developer kit, which measures the response time for each request in real time application. The average response time for each implementation is calculated and compared with its counterpart.

A. *Simulation*

"In its narrowest sense, a computer simulation is a program that is run on a computer and that uses step-by-step methods to explore the approximate behavior of a mathematical model. Usually this is a model of a real-world system." [11]

In [12], author states the following key considerations for software simulations:

- *Timing*: Timing is an important factor to take into consideration when simulations are involved. By considering the timing factor, it is much easier to evaluate the effect of the developed system on different parts of the whole system.
- *Matching to Reality*: In order to get as realistic results as possible, it is important to get the simulations to work in scenarios as close as possible to the real-world ones, although in most cases this might not be fully achievable due to the limitations of the testing environments.
- *Effective Testing*: Selection of the testing methods is crucial in simulations. It is important to take into consideration different factors, such as the environment in which the testing is being done together with the

right testing methods, to get the results as precise as possible.

B. Using Instant Messaging System as a Simulation for Real-Time Applications

There are numerous real-time applications but possibly the most well-known are the chatting software. Chatting applications are among the most popular and typical real-time applications which are still widely used in the market. Chatting applications nowadays are not limited to the text messages only. The response time for these applications is almost instant, but there are various factors involved in the performance of these applications that include network traffic, latency and for the high graphical contents, the system memory. Since instant messaging applications, are well-known and common, they are perfect to be used for the real-time simulations, as these are as pretty close to the real world scenarios.

Due to the popularity of these applications, they are great to be used in real-time simulations. They are particularly good for the purpose of this research, as each design pattern has its own unique usage in these applications. For instance, the State design pattern can be used to play the multimedia content in these applications, the pattern can be helpful to determine the status of the media player.

C. Use of Design Patterns in the Simulated Application

For the purpose of this research, we developed an instant messing application, in which the three discussed designed patterns (Observer, State and Strategy) could be implemented. By estimating the response time using each of these patterns as well as their equivalent simpler solutions, the results are easily comparable.

Observer design pattern is used to keep track of certain objects and provide response to the changes that might happen to them. The idea is to think of the Observer object as an event, which occurs while the chatting system is working, this event can be a right click, left click, or any other event. The system works in a way that it repeatedly changes the status of the objects and prints them to the browser's console. The same behaviors are coded using its simpler solution (without using design patterns) so that the difference in the performance of application can clearly be seen on console.

State design pattern is used to simulate the media player that has three states, play, pause and stop. Like the previous design pattern, once the program is running on its periodical way, the state will continuously change from one to another, and the results will be printed out to the console of the web browser. The equivalent simpler version of the application achieves the same results.

Strategy design pattern is applied to simulate the use of emoji in the chat application. The emojies are displayed based on certain types of string inputs. The idea behind using Strategy pattern here is to minimize the code complexity by having separate methods for different types. Here the simulation is designed in a way that it will repeatedly printout the wordings for three different signs, using Strategy design pattern. Similarly its equivalent simpler solution is implemented without the use of design patterns.

D. Simulation Environment

As mentioned before that the result of each simulation is strongly under influence of the environment in which the simulation is done. Changing the environment of the application may affect the results of the simulation. Table 1 demonstrates the environment in which the simulation is performed.

TABLE I. SYSTEM SPECIFICATIONS

OS	**HDD**	**CPU**	**Memory**	**Time Interval**
Windows 8.1, 64 bit	SATA	Intel i3, 2.7 GHz	4 GB	200 ms

E. Performance Measurement

The program simulates a simple chatting application where by typing in the chat window and clicking on the send button, the application will send the message. In order to run the simulation, the application is designed to send a message continuously on a specific interval that has been set for this purpose. This way a certain word or phrase will be typed in the message box and send to the application continuously. In order to do so, following steps are followed:

- Enter a word or phrase in the messaging console and click outside of the box (focus out) so that the program captures the word or phrase.
- Set the time interval (default setting is 200 milliseconds), select the checkbox and click on continuously running.

By following the above steps, the program continuously send the saved word or phrase to the console on the given time interval.

By using Google Chrome's Developer Tools, the results of the simulation can be seen. In order to activate the Google Chrome's Developer's Toolkit, the below steps should be taken:

- Right click on the browser's screen and choose "Inspect".
- Select Console tab.

We need to make sure that Google Chrome's timer option is activated. In order to activate the timer, in Developer's Toolkit, click "Console settings" and check the option "Show timestamps".

Google Chrome's timestamp setting shows the response times up to three digits in milliseconds, which is good enough to demonstrate the speed difference between different types of implementations.

F. Challenges and Constraints

There are some challenges and constraints when it comes to test the performance of the application.

- One of the major challenges is to produce enough loads of data to show the performance difference between different implementations of the code, as the difference might not be that great in small data loads, and it will only show up in heavy traffic.

- As there are many different factors that might affect the results of the test, it is important to ensure that factors such as the network speed, hard-disk rate remains the same throughout the entire testing period.
- Another challenge is the small factor of change shown in the system performance for every request. It is important to mention that although the effect of the used design patterns might be minimal in a small scale, however at large scales, it will be significant.

IV. Findings

In this research, the performance of the codes for each design pattern is tested using the designed simulation. The time difference between the responses times are the unit of measure for testing the performance of each code. Fig. 4 demonstrates the results of running the State design pattern timestamps. By applying the below calculation to every pair of timestamps, the minimum and the maximum values can be calculated and by dividing the accumulative values of the calculation to the number of pairs, the average result can be identified.

For example how the time response for a single send and receive can be calculated by using above timestamps is illustrated in Fig. 5. These values represent the first pair of state design pattern timestamp results.

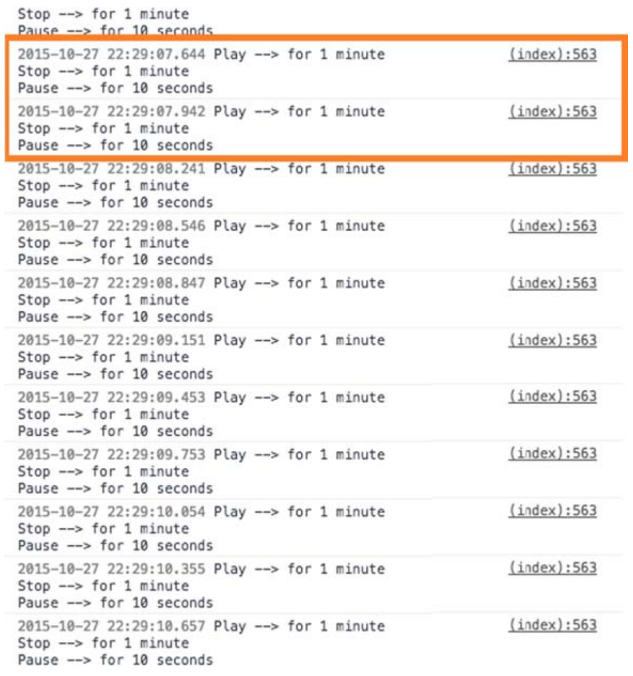
The time difference between two requests can be calculated as follows:

- The response time from the first request: 22:29:07.644
- The response time from the second request: 22:29:07.942
- Time difference: 07.942 - 07.644 = 0.298 ms

Therefore, the time difference between two requests is about 0.298 milliseconds.

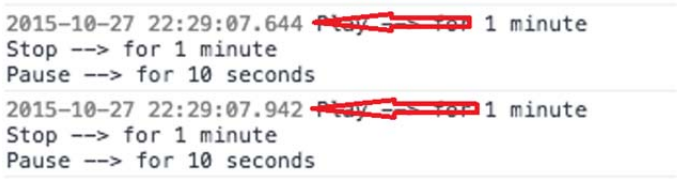
TABLE II. State Pattern: Pair Results

Pairs	Upper Value	Lower Value	Results
1st	07.644	07.942	0.298
2nd	07.942	08.241	0.299
3rd	08.241	08.546	0.305
4th	08.546	08.847	0.301
5th	08.847	09.151	0.304
6th	09.151	09.453	0.302
7th	09.453	09.753	0.300
8th	09.753	10.054	0.301
9th	10.054	10.355	0.301
10th	10.355	10.657	0.302



```
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:07.644 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:07.942 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:08.241 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:08.546 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:08.847 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:09.151 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:09.453 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:09.753 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:10.054 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:10.355 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:10.657 Play --> for 1 minute            (index):563
Stop --> for 1 minute
Pause --> for 10 seconds
```

Fig. 4. Timestamps for state design pattern based solution.



```
2015-10-27 22:29:07.644 ~~Play --> for~~ 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:29:07.942 ~~Play --> for~~ 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
```

Fig. 5. First two timestamp for state design pattern.

TABLE III. State Pattern: Overall Results

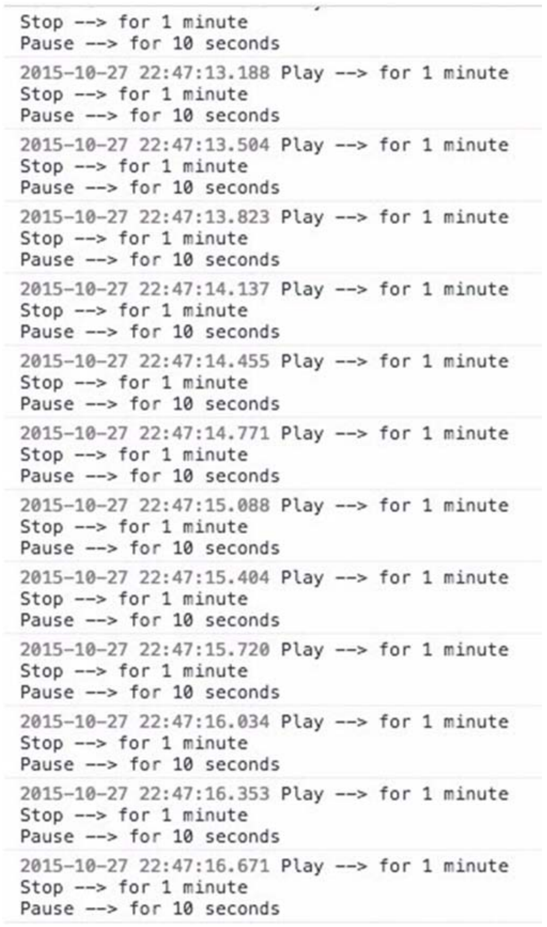
Status	Interval	Min	Max	Average
successfully run	200	298	305	302

Table 2 above shows the State design pattern pair results whereas Table 3 shows the Minimum value (which is the least value in the pairs of state design patterns timestamps results), Maximum value (which is the highest value in the pairs of state design patters timestamps results) and the Average (which shows the average value of the total pairs of the state design patterns timestamps results). Interval is fixed for all the simulations.

Later we implemented the same functionality using the traditional coding methods without implementing the State design pattern. Fig. 6 demonstrates the performance of the equivalent solution without using the State design pattern.

Table 4 shows the State equivalent simpler solution pair results whereas Table 5 shows the calculated Minimum value, Maximum value and the Average. Interval is fixed for all the simulations.

In Fig. 7, the bar chart represents the timestamps results of state design patterns compared to the timestamps results of its equivalent simpler solution.



```
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:13.188 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:13.504 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:13.823 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:14.137 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:14.455 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:14.771 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:15.088 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:15.404 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:15.720 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:16.034 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:16.353 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
2015-10-27 22:47:16.671 Play --> for 1 minute
Stop --> for 1 minute
Pause --> for 10 seconds
```

Fig. 6. Timestamps for equivalent simpler solution without using state design pattern.

TABLE IV. STATE EQUIVALENT SIMPLER SOLUTION PAIRS

State Simpler Equivalent Pairs	Upper value	Lower Value	Results
1st	13.188	13.504	0.316
2nd	13.504	13.823	0.319
3th	13.823	14.137	0.314
4th	14.137	14.455	0.318
5th	14.455	14.771	0.316
6th	14.771	15.088	0.317
7th	15.088	15.404	0.316
8th	15.404	15.720	0.316
9th	15.720	16.034	0.314
10th	16.034	16.353	0.319

TABLE V. STATE EQUIVALENT SIMPLER SOLUTION: OVERALL RESULTS

Status	Interval	Min	Max	Average
Successfully run	200	314	319	316

TABLE VI. DESIGN PATTERN BASED SOLUTION VS. SIMPLER EQUIVALENT

	Interval	Min	Max	Avg.
State	200	298	305	302
State simpler equivalent	200	314	319	316
Observer	200	296	305	302
Observer simpler equivalent	200	306	312	307
Strategy	200	314	318	317
Strategy simpler equivalent	200	313	318	316

Similarly the values for Observer design pattern and Strategy design pattern were calculated. Table 6 demonstrates the performance rate of each design pattern, compared to its equivalent code.

Table 6 demonstrates that State design pattern increases the performance rate by average 0.012 seconds compared to its traditional equivalent code. By using the Observer design pattern, we also managed to get improved results as compared to its equivalent simpler solution, which is an average of 0.005 seconds higher in performance. But by using the Strategy pattern, the effect of the pattern on the performance application code seems almost none (about 0.001 only). However the Strategy design pattern is primarily meant to increase the code readability and enhancing the development process, rather than affecting the performance of the application.

Fig. 8 and 9 represent the comparison of timestamp results of Observer and Strategy design pattern based solutions with their equivalent simpler solutions.

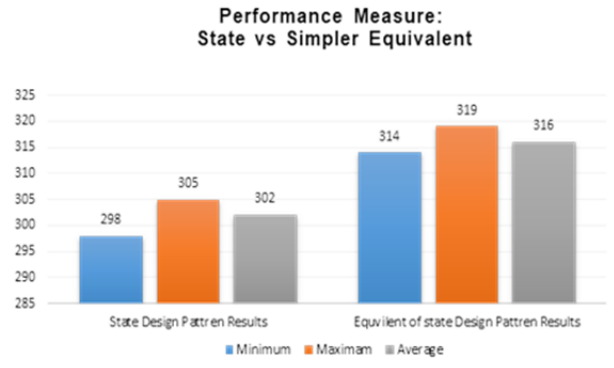


Fig. 7. State results vs. Its simpler equivalent results.

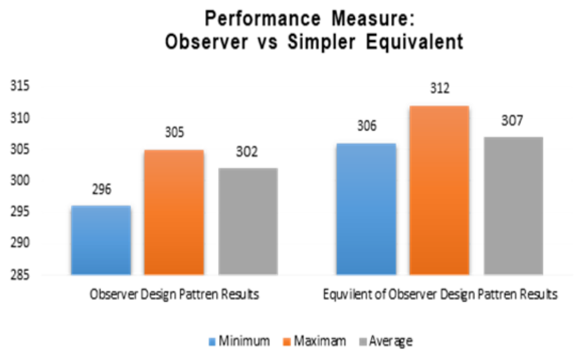


Fig. 8. Observer results vs. Its simpler equivalent results.

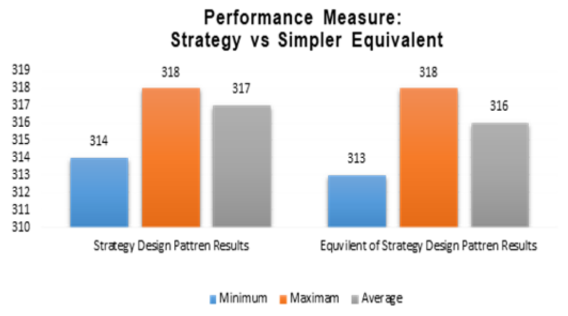


Fig. 9. Strategy results vs. Its simpler equivalent results.

V. Conclusion

Due to the increasing demand for fast-responding applications, real-time applications play an important role in current IT industry. There has been attempts to enhance the performance of these type of applications by using various solutions, namely hardware and software optimization. However as software optimization is less tangible, there is a dire need for further research in this area to find out more suitable and effective methods. The current research is an attempt to study the importance of design patterns in order to improve the performance of real-time applications. Two of the design patterns used by the researchers, namely the Observer and State design patterns, proved to be more effective in terms of software efficiency.

This research provides the basis for future work on this topic which might involve an assessment of other design patterns as well as judging their performance on various other real-time application scenarios.

References

[1] Hein Pattyn. (2015). 3 Keys to Improving Enterprise Application Performance and the Business User Experience. Available: http://www.infovista.com/2015/09/10/3-keys-to-improving-enterprise-application-performance-and-the-business-user-experience/. Last accessed 05/09/2015.

[2] Lisa Phifer. (2012). Optimizing Network Performance for WLAN Real-time Applications. Available: http://searchnetworking.techtarget.com/tip/Optimizing-network-performance-for-WLAN-real-time-applications. Last accessed 05/09/2015.

[3] John Huntington. (2006). Real Time. Available: http://whatis.techtarget.com/definition/real-time. Last accessed 09/01/2015.

[4] Stephen Gilmore (2013). Trends in Functional Programming, Volume 4. Chicago: University of Chicago Press. 2-3.

[5] EventHelix.com Inc. (2015). Issues in Real-time System Design. Available: http://www.eventhelix.com/RealtimeMantra/IssuesInRealtimeSystemDesign.htm#.VgA1G9-qqkp. Last accessed 05/09/2015.

[6] Margaret Rouse. (2007). Pattern (Design Pattern) Definition. Available: http://searchsoftwarequality.techtarget.com/definition/pattern. Last accessed 12/09/2015.

[7] Ali, M. & Elish, M. O. (2013). A Comparative Literature Survey of Design Patterns Impact on Software Quality, p.1-7.

[8] Jeff Friesen . (2012). Design patterns, the big picture, Part 1: Design pattern history and classification. Available: http://www.javaworld.com/article/2078665/core-java/design-patterns--the-big-picture--part-1--design-pattern-history-and-classification.html. Last accessed 09/01/2015.

[9] Gamma, E., Helms, R., Johnson R. & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley.

[10] Nat Hillary. (2005). Measuring Performance for Real-Time Systems.Freescale Semiconductor, 14 (2), 3-5.

[11] stanford.edu. (2015). Computer Simulations in Science. Available: http://plato.stanford.edu/entries/simulations-science/#WhaComSim. Last accessed 10/02/2015. Last accessed 10/02/2015.

[12] simul8. (2015). What is Simulation?. Available: http://www.simul8.com/products/what_is_simulation.htm. Last accessed 03/10/2015.