A design pattern for real time progress monitoring of a time-intensive process in a stateless software application

Md. Mohaimenuzzaman*

United International University,
House # 80, Road # 8/A,
Dhanmondi, Dhaka-1209, Bangladesh
Email: mohinr@gmail.com
*Corresponding author

S.M. Monzurur Rahman

Samsung R&D Institute Bangladesh,
111, Bir Uttam C. R. Dutta Road,
Dhaka-1205, Bangladesh
Email: rahman.sm@samsung.com
Email: smrahman99@gmail.com

Abstract: As stateless software applications run based on request and response to and from a server, it depends on the server processes to manage concurrent requests and its session state. In the cases, where there are modifications to the session state, the server will have to handle the requests synchronously to prevent many issues relating to race conditions. The server components usually have a maximum execution timeout for requests to finish its execution. It is possible to extend this limit but it may open a security threat for the application as it is now more susceptible to denial of service attacks. With these limitations, we propose a pattern to allow for time-intensive processes to run in a stateless application and the ability to monitor their progress in real time with other capabilities such as stopping, pausing and resuming the background task.

Keywords: Ajax; asynchronous processing; design pattern; multi-threading; session; stateless application; uninterrupted process; threading; web application; web server.

Reference to this paper should be made as follows: Mohaimenuzzaman, M. and Monzurur Rahman, S.M. (2015) 'A design pattern for real time progress monitoring of a time-intensive process in a stateless software application', *Int. J. Software Engineering, Technology and Applications*, Vol. 1, No. 1, pp.53–63.

Biographical notes: Md. Mohaimenuzzaman is working as a freelance Software Engineer for the last six years. He is currently working as a Software Engineer at PureHome Corporation, Seattle, WA 98122. He received his BSc in Computer Science and Engineering from Eastern University, Bangladesh in 2007 and MSc in Computer Science and Engineering from United International University, Bangladesh in 2013. He also received the Best Freelance Software Developer 2012 award from Bangladesh Association of Software and Information Services (BASIS) in 2012.

S.M. Monzurur Rahman is currently working as the Director of Samsung Research and Development Institute in Bangladesh. Prior to joining Samsung, he served as a Professor of Computer Science and Engineering for United International University, Bangladesh. He received his BSc in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Bangladesh in 1993, MSc in Machine Learning from Central Queensland University, Australia in 1996 and PhD in Computer and Electrical Engineering from RMIT University, Melbourne, Australia in 2006. He has a number of published research papers to his credit in internationally reputed journals. He is also the author of the book *Data Mining Using Neural Networks*. He is a member of the editorial board of *International Journal of Data Science (IJDS)* (http://www.inderscience.com/jhome.php?jcode=ijds#edboard).

1 Introduction

For a system to report the progress to a client about a long running process on the server, the client and the server need to be in continuous communication. In desktop applications the user interface is always connected to the background task, it is straightforward to monitor this task and report this progress to the user. Nevertheless, the scenario is not the same for stateless software applications. In stateless environment, the server treats every request independently and does not retain information of any previous connection (Mein et al., 2002). As HTTP is a stateless protocol (Evjen et al., 2010; Offutt and Wu, 2010) and web-based applications and cloud-based applications run over HTTP, they are stateless by default. Although it is possible to implement them as stateful, this paper only deals with stateless applications. To reduce the scope of this paper, we will only focus on web-based applications.

Due to the stateless nature of the web, the browser page which acts as a client, is not always connected to the code running at the server. It runs based on a request and response model, which means the client stays connected to the server just long enough to get the response back from the server before it disconnects (Tian and Jun, 2013). Because of this limitation of the HTTP protocol the server cannot push data to client on its own (Ying et al., 2013). On the other hand, server technologies have a default execution timeout for a request to finish its execution (e.g., 90 seconds for asp.net and 30 seconds for php) (Evjen et al., 2010; Lerdorf et al., 2002a). Therefore, a process that takes more than the default execution timeout experiences a time out exception and the client receives a request time out error in the browser. Although there are means to extend the default execution timeout limit via server configurations, doing it increases the application's susceptibility to a denial of service attack (Meier et al., 2003).

Looking further into this process, we found that web servers impose an exclusive lock to the session state while processing a request to maintain its integrity (Lerdorf et al., 2002b; ASP.NET Session State Overview, 2013). Therefore, if concurrent requests are made from the same user session, the first request gets exclusive access to the session information and until the first request completes its execution and frees session lock, the second request does not execute (ASP.NET Session State Overview, 2013). This makes it clear that while keeping the session state in write mode it is not possible to use the session as a mechanism for communication among parallel requests.

In this circumstance, it is very important to have a common solution that will help web-based application developers to implement time-intensive processes with the ability to monitor their execution. This paper aims to develop a solution using a number of different available techniques together to overcome the design problem that would enable a time-intensive process to complete its execution regardless of the default execution timeout. In addition, the solution should also allow for progress monitoring of the time-intensive process and also ensures access to the session information from parallel processes. For the purposes of feature parity with traditional applications, the solution should allow pausing and termination of the time-intensive process.

We propose web server process progress monitor (WPM), a design pattern for such problems in web-based application development and we categorise this pattern as behavioural patterns. Before presenting the WPM design pattern in details, we will cover some commonly practiced approaches as related works in Section 2. Section 3 describes the proposed web design pattern including the client server communication, use case diagram, class diagram, consequences and implementation that illustrates how different features of the proposed design pattern may be implemented. Section 4 concludes our paper.

2 Background

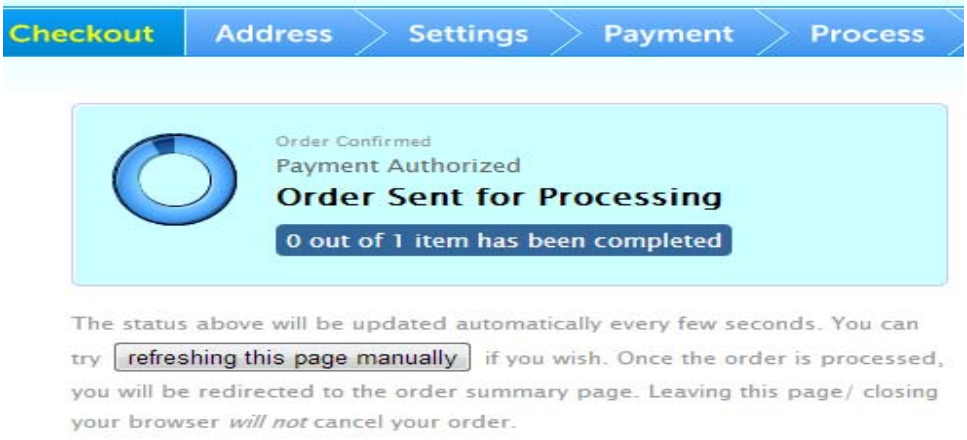
Modern web application constructions have features that often require a lengthy period for processing. It has become important to have a real-time monitor that shows the progress of the process and provides the user with ability to pause or terminate the running process to make the feature more interactive. Therefore, to have a progress monitor, the running process needs to store its progress somewhere that is easily accessible by other processes. To let the user know what actually is happening on the server, the client has to read the progress periodically after launching the time-intensive process and repeats until the process finishes.

With the limitations in traditional web model, there are some approaches being practiced currently to accomplish the complex tasks in web-based applications. In one of the approaches it submits the request to the server using a full page post and redirects the user to a new page where a static message or an animated image is displayed indicating that the task is in progress. In this approach, the user never knows about the happenings in the background. The IEEE Graphics Checker Tool uses a similar approach for its graphics validation process.

To provide the user a better experience, Ajax (asynchronous JavaScript and XML) requests can be used. Ajax requests are used to call server side functions asynchronously as these requests are lightweight and are generally preferred over the traditional complete post back (Dhand, 2011). Tian and Jun (2013) proposed in their paper that the advantage of Ajax can be taken to send asynchronous HTTP requests to server to process complex tasks which also provides the user the ability to continue working with the same page. In addition, there may be other subsequent HTTP requests using Ajax to provide the user a better ability to know the progress of the process. If we look at payment processing systems in web-based applications we find that multiple tasks are accomplished to complete an online order. Although these types of processing are not always time consuming, multiple tasks make them important in this context. We might also see the page gets refreshed in regular interval with new progress status. One important point to

notice here is that once the process is started, user cannot control it. Moreover, to load the new status the whole page needs to reload. Figure 1 shows a payment processing scenario in real time.

Figure 1 Payment processing system with user level instruction (see online version for colours)



Although the above approaches are straight forward, they fail when they are unable to complete the process execution under the normal timeout. In a typical web application, most processes that executes on the server will reach completion before the normal timeout is reached. Having this normal timeout increases the reliability of the system because it also serves the role of stopping a process that is in trouble and frees up resources to keep the server running. Altering this default normal timeout for all calls to accommodate the time-intensive process removes this important system control facility. Moreover, extending the timeout does not always work since the time for a process to finish its execution could be non-deterministic. In addition, concurrent requests sent to the server to launch the process and gathering progress of the running process cannot use the session state simultaneously. As session state is not available, we need to consider alternative storage mechanisms that may not be feasible. The described mechanisms do not afford the possibility of pausing or terminating the running process.

In this context we may look at using asynchronous mechanism for complex and time consuming processes. Resch et al. and Zhao et al. suggested this mechanism for complex and performance complicated algorithms that take longer than the usual timeout which indeed gives the clients the ability to stay responsive and continue their processing without waiting for the response (Resch et al., 2010; Zhao et al., 2012). Asynchronous execution of the time-intensive process resolves the problem of server and browser timeouts and it also provides the opportunity of sharing information across multiple requests that keeps the system simple and performant. We chose to focus on minimising the lock time on session state to achieve the same result while increasing performance.

To display process execution progress in the client, the progress needs to be retrieved continuously from server. Using SOCKET, a HTML5 feature, the server can push data instantly to the browser (Pohja, 2010; Chen and Zhao, 2010). However it has compatibility issues since all browsers do not support this feature. Therefore, subsequent

HTTP requests using Ajax might be the best solution for continuous status polling from server (Ying et al., 2013).

Considering the above discussions and related works, we see that the time-intensive process can be run as an asynchronous process. For retrieving continuous progress from server to display the progress in the client, we chose to stay with Ajax which does not have any compatibility issues with browsers. Finally, having the ability to pause and terminate the running process improves the user experience greatly. We consider these requirements together as a common design problem that drove us to come up with WPM, WPM which provides a straight forward common technique of implementing time-intensive process in web-based applications with the ability to monitor the execution of the time-intensive process in real time. In the next section we will present WPM in details.

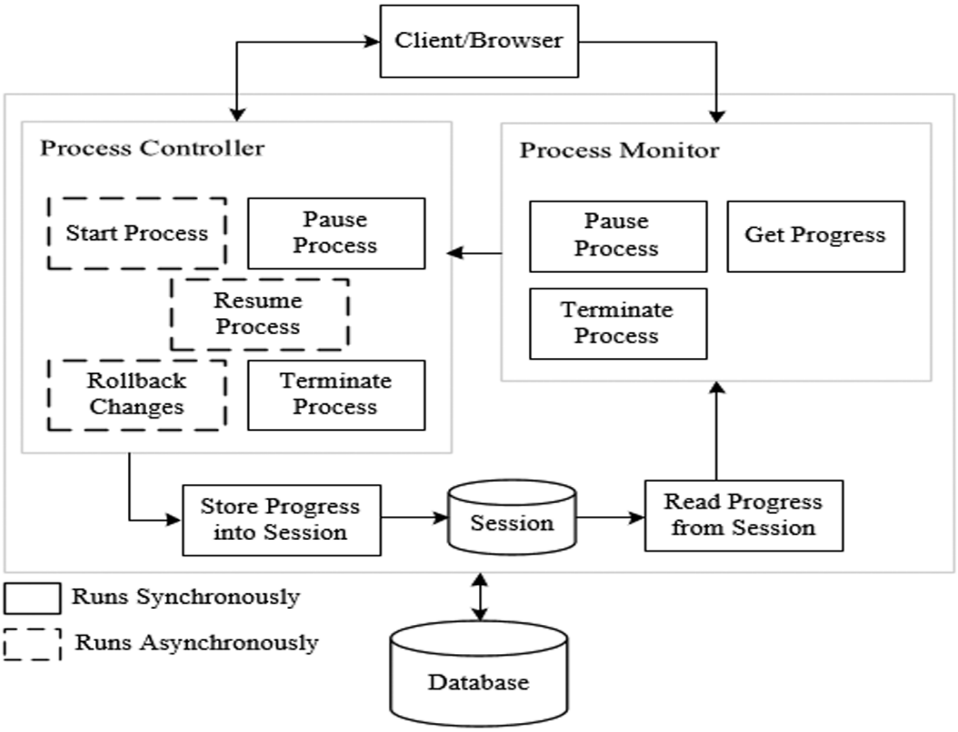
3 Web server process progress monitor (WPM)

From the discussions in Section 2, we have found that to implement a real time progress monitor of a time-intensive process in web based applications we need to make the following things possible:

- complete the time-intensive process execution regardless of the web server's default execution timeout
- the client request for time-intensive process execution needs to complete its lifecycle without waiting for the completion of execution
- the time-intensive process needs to run in such a way that it can be paused or terminated anytime
- the client needs to have the continuous progress status to display it until the process completes its execution.

We have taken all of the above into consideration in WPM. The core idea of WPM is to let the time-intensive process running asynchronously in the background on the server to allow the request to complete its lifecycle immediately without waiting for the time-intensive process to finish which ensures that it is no longer a blocking call to the server. This also prevents the session state from being locked during the execution of the long-running process which enables other requests to be processed by the web server without having to wait to acquire a lock on the session state. Subsequent HTTP requests with the help of Ajax are made to server to retrieve the progress status and the server processes the request immediately by retrieving the current progress from the session. If the user wants to pause or terminate the time-intensive process execution, a separate HTTP request using Ajax is sent to the server with an appropriate flag to indicate termination. This flag tells the server to abort the execution of the background process and stores the necessary information so that the operation can be resumed at a later time. At completion or at termination point of the time-intensive process, all the progresses can be stored permanently so that the history of the life-time of a process can be accessed later.

Figure 2 Communication between components of WPM



WPM has two main components – the *process controller* and the *process monitor*. Figure 2 shows the communication between them. The *process controller* is responsible for launching the time-intensive process, pausing or terminating a running process, resuming a paused process, or rolling back any incomplete process that the user does not want to complete. Upon a user's request, the *process controller* starts an asynchronous operation in the background and allows the request to complete its lifecycle immediately after the execution starts. The *process controller* handles resume and rollback operations in the same way. To provide resume or rollback capability, the process controller checks if there are any incomplete processes. If any are found, the user is prompted with choices to complete the process or rollback the changes.

Process monitor is responsible for receiving client requests for current progress, pausing or terminating process execution. *Get progress* handles the request for getting progress status which then forwards this request to *read progress*. Once the client gets confirmation from the *process controller*, usually in the form of a 200 HTTP response code that it has launched the time-intensive process, it starts polling *process monitor* for progress. If *process monitor* receives a pause or terminate request, it calls *process controller* to do the operation. To give a better sense of how this model works, a use case diagram is shown in Figure 3 and finally we present WPM design pattern using class diagram in Figure 4.

Figure 3 Use case diagram for WPM

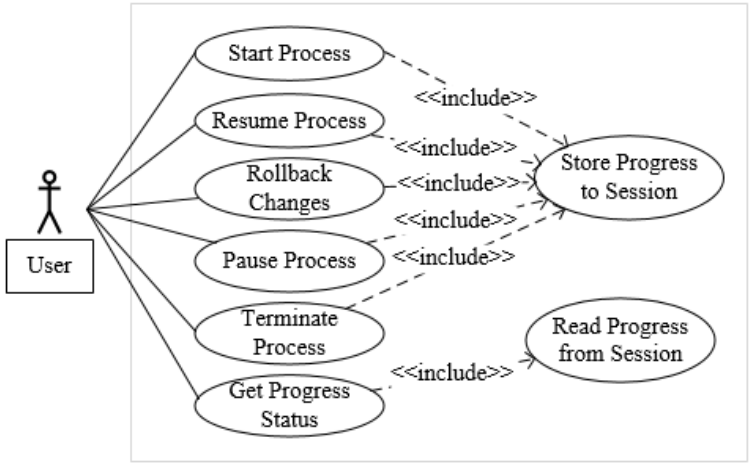
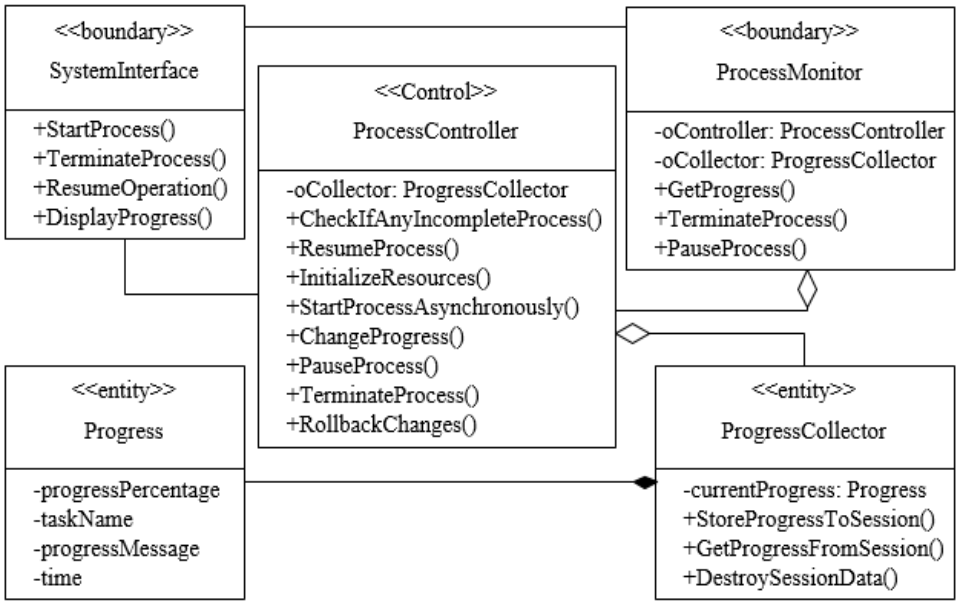


Figure 4 Web server process progress monitor



3.1 Class diagram for WPM

A number of classes together bring WPM into reality. Figure 4 demonstrates those classes, their responsibilities and the relationships between them.

3.2 Collaboration between participant classes

Now we discuss about the responsibilities among the participant classes in WPM.

1 Progress
   - defines the progress object of the time-intensive process.

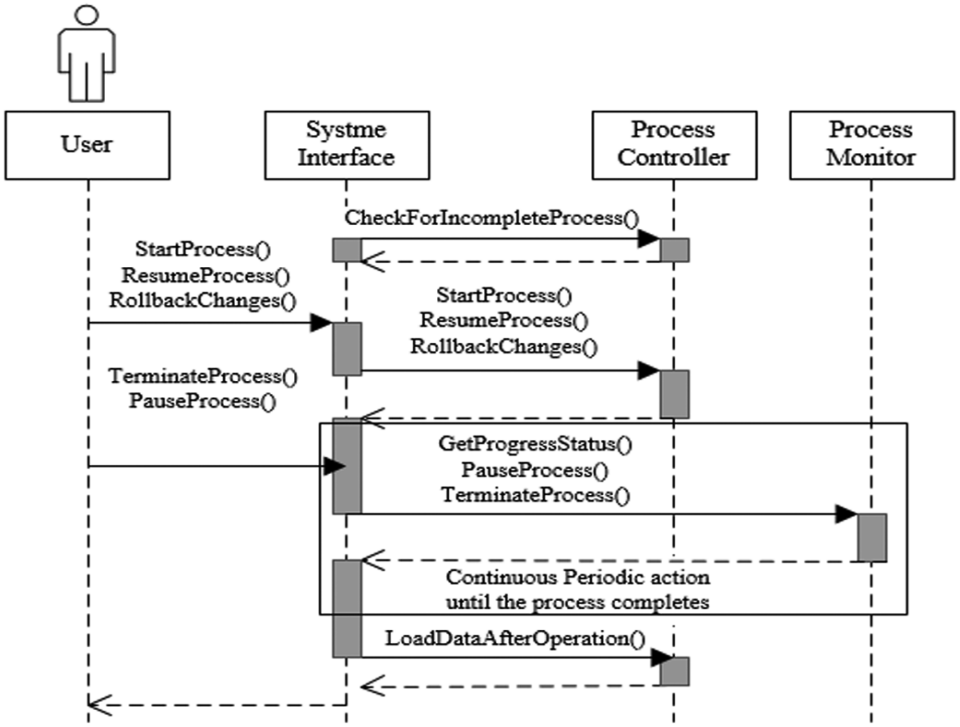
2 ProgressCollector
   - writes a progress object to the session
   - reads a progress object from the session
   - destroys progress data from the session.

3 ProcessController
   - checks if there are any incomplete processes to resume or rollback
   - executes the time-intensive process, resume the process, or rollback changes
   - stores process progress to the session with the help of ProgressCollector.

4 ProcessMonitor
   - gets the progress using ProgressCollector and sends it back to SystemInterface
   - requests ProcessController to pause or terminate the running process.

5 SystemInterface
   - sends start, resume or rollback operation request to ProcessController
   - sends periodic HTTP requests using Ajax to ProcessMonitor for progress
   - sends process termination or pause request to the ProcessMonitor.

Figure 5 Interaction between participant classes in WPM



For a better visualisation of the collaboration between the participating classes of WPM, an interaction diagram is shown in Figure 5.

3.3 Consequences

- *The time-intensive process execution is no longer a blocking call:* Since the server no longer waits for the time-intensive process to finish, it returns immediately to the client and thus closing the connection between the client and the server.
- *No need to extend the default execution timeout:* The time-intensive process runs asynchronously in the background and the request completes its lifecycle within the default execution timeout of the web server. The request is no longer responsible for the execution of the process, it merely asks the server to launch the process and run it in the background.
- *No session lock:* The time-intensive process is now a background process and the response is sent back to the client immediately after ProcessController launches the process, the session state is no longer in a write-lock.
- *Real-time progress status:* Subsequent HTTP requests with the help of Ajax can be sent to server for progress status. Since the user session is not in a write-lock, the server processes other requests immediately and sends the response back to the client.
- *Possible to terminate, pause or resume a time-intensive operation:* An asynchronous process generally runs in the background within a separate thread. Therefore, it can be terminated anytime by sending a separate HTTP request to ProcessMonitor. Pause is one kind of termination, but by keeping extra information around so that it can be started from the same point again.
- *No concern of session size limitation:* For faster read and write, progress objects are stored in session and the space needed for this purpose is negligible.
- *Possibility of deadlock in multi-threading environment is negligible:* When a separate thread is used for asynchronous execution of the time-intensive process, it becomes multi-threading (SunSoft, 1994). When there are dependencies between multiple threads, if one of them hangs because of an error or for any other issue, a deadlock may occur (Agarwal et al., 2010). In case of WPM, no other thread depends on the thread that is used for asynchronous processing. Therefore the possibility of a deadlock is negligible in WPM.
- *Concern about session conflict:* There may be concerns about session conflict when a user starts multiple time-intensive processes from the same client. Session identification mechanism can be used easily to remove this concern.

3.4 Implementation

Here are some useful techniques to effectively implement a time-intensive process and a monitoring system for it by following WPM design pattern.

- *Implementing time-intensive process:* The execution of time-intensive process needs to be asynchronous and needs to be able to write the progress continuously into the session so that other processes can access it. This means the calling method returns before the task finishes and the page can complete its request lifecycle.
- *Implementing pause or terminate feature:* The thread component under which the time-intensive process execution happens needs to be accessible to control its execution.
- *Implementing resume/rollback feature:* ProcessController needs to have a mechanism to check if there are any incomplete processes that was previously paused. If found any, the user can be prompted with options whether he wants to resume it or rollback the changes.
- *Reading progress status:* Since the session state is accessible by other processes and ProcessController stores the progress object into session, the client may have the continuous updates on progress by making repeated HTTP request with the help of Ajax to ProcessMonitor.

We have placed our sample code at http://www.mmohin.com/articles/wpm/sample-code.html..

4 Conclusions

The implementation of a real-time progress monitor in a web application is difficult since in web, the client is not always connected to the server. Moreover, web servers do not process concurrent requests to maintain the integrity of user session. In addition, extending default timeout of a web-server to accommodate a time-intensive process removes important security control from the application. Our proposed design pattern – WPM – addresses all of these design issues. It allows the time-intensive process to run in the background asynchronously without blocking the connection during the execution period of a time-intensive process. This enables other requests to be executed with exclusive access to session information along with availing pause, resume or termination features.

WPM currently focuses on web-based applications only. There remains immense scope for further work to make the design pattern applicable for cloud-based applications and mobile applications that use HTTP protocol for processing data in server side. The concept of Web Socket instead of Ajax might be another piece of work in the future that may help WMP more performant. Since WebSocket comes with HTML5 and later, we consider it as a further scope of work for WPM.

References

Agarwal, R., Bensalem, S., Farchi, E., Havelund, K., Nir-Buchbinder, Y., Stoller, S. D., Ur, S. and Wang, L. (2010) 'Detection of deadlock potentials in multithreaded programs', *IBM J. Res. & Dev.*, Vol. 54, No. 5, pp.1–15.

ASP.NET Session State Overview (2013) [online] http://msdn.microsoft.com/en-us/library/ms178581%28v=vs.100%29.aspx (accessed 7 November 2013).

Chen, H. and Zhao, F. (2010) 'Implementation of web instant message system based on server push technology and XMPP', *Computer Engineering and Design*, Vol. 31, No. 5, pp.83–88.

Dhand, R. (2011) 'AJAX based web services invocation: higher performance & bandwidth utilization', *2009 International Conference on Machine Learning and Computing*, IPCSIT, Darwin, pp.2–5.

Evjen, B., Hanselman, S. and Rader, D. (2010) 'ASP webserver controls', *Professional ASP.NET 4 in C# and VB*, pp.457–489, Wiley Publishing, Inc., Indiana.

Lerdorf, R., Tatroe, K., Kaehms, B. and McGredy, R. (2002a) 'Appendix A: function reference', *Programming PHP*, p.439, O'Reilly & Associates, Inc, California.

Lerdorf, R., Tatroe, K., Kaehms, B. and McGredy, R. (2002b) 'Application techniques', *Programming PHP*, pp.297–316, O'Reilly & Associates, Inc, California.

Meier, J.D., Mackman, A., Vasireddy, S., Dunner, M., Escamilla, R. and Murukan, A. (2003) 'Securing your ASP.NET application and web services', *Improving Web Application Security: Threats and Countermeasures*, pp.543–588, Microsoft Corporation, Redmond, Washington.

Mein, G., Pal, S., Dhondu, G., Anand, T.K., Stonjanvic, A., Al-Ghosein, M. and Oeuvray, P.M. (2004) *Simple Object Access Protocol*, U.S. Patent No. 6782542, U.S. Patent and Trademark Office, Washington, DC.

Offutt, J. and Wu, Y. (2010) 'Modeling presentation layers of web applications for testing', *Software and System Modeling, Springer*, Vol. 9, No. 2, pp.257–280.

Pohja, M. (2010) 'Server push for web applications via instant messaging', *Journal of Web Engineering*, Vol. 9, No. 3, pp.227–242.

Resch, B., Sagl, G., Blaschke, T. and Mittlboeck, M. (2010) 'Distributed web-processing for ubiquitous information services – OGC WPS critically revisited', *Proceedings of the 6th International Conference on Geographic Information Science (GIScience2010)*, pp.14–17.

SunSoft (1994) *Multithreaded Programming Guide*, pp.2–10, Sun Microsystems, Inc., Mountain View, California.

Tian, L. and Jun, W. (2013) 'Design and implementation of instant communication systems based on Ajax', *Proceedings of the 2012 International Conference of Modern Computer Science and Applications,* Springer Berlin Heidelberg, pp.687–693.

Ying, G., Lei, M., Zhonghu, H. and Weiyang, Z. (2013) 'An architecture to implement event-driven web monitoring systems', *TELKOMNIKA Indonesian Journal of Electrical Engineering*, Vol. 11, No. 10, pp.6068–6073.

Zhao, P., Foerster, T. and Yue, P. (2012) 'The geoprocessing web', *Computers & Geosciences*, Vol. 47, pp.3–12.