WEB SOCKETS ENABLED ON-LINE PRESENCE AND REALTIME GAME NAVIGATION IN MONOPOLY CLOUD SOLUTION

Nino Karas, Stefan Mitev, Marjan Gusev, Dragan Sahpaski
Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Skopje, Republic of Macedonia
{stefan.mitev, nino.karas, marjan.gushev, dragan.sahpaski @finki.ukim.mk}

ABSTRACT

The project goal was to create a Software as a Service - cloud solution for the Monopoly game, where a lot of independent entities (students) will build web services and enable realization of a real-time game. In the process of realization and implementation we have faced two big challenges: web service orchestration and real-time multiplayer game navigation. The aim of this article is to present how we faced and solved the multiplayer navigation challenge using the latest IT technologies. In this paper we will give a description of the implementation and why we chose the HTML 5 web sockets as a core technology for the solution over the other known technologies.

I. INTRODUCTION

The number of active on-line users is increasing along with global expansion of Internet. To cope with increased demand, the IT scientists and developers work hard to develop more powerful and sophisticated technologies. As a consequence, the Internet applications and services, supported by these new technologies become more complex and new problems arise concerning the Internet services and their integration in many daily used applications. For example, the on-line presence of the user has to be solved for different entities in cloud since it has become a crucial part of almost every modern Internet application. Besides the online presence, the requirements of the modern Internet users became more complex as today they are looking for complete solutions of their problems in the cloud.

Game industry as increasing trend set also a lot of new requirements. For example, the users are looking to play online multiplayer games that are completely in the cloud and don't require any additional software parts to be installed on the user's computer.

We developed and implemented a complete cloud solution of a game that allows users to play an extended multiplayer version of the classic monopoly board game. The game extends the classic board game in a way that the theme – story of the game is changed and some additional rules are added based on the story. Our version of the monopoly game uses a map of all Macedonian cities and predefined route to cross them, more complex banking system, more complex procedure to buy and build objects, some additional features depending on which city the user is currently visiting etc.

Implementation of these additional features required completely separate web services that communicate between each other. We have also set a goal that the game will be played in multiplayer environment and that it will feature real time navigation of the players and update the game statistics. While implementing the game we face two big challenges: the orchestration of many independent web services, and real time multiplayer game navigation. In order to solve the real-time navigation we have performed several experiments using the latest IT technologies.

II. PROBLEM DEFINITION

By analysing the classic Monopoly board game, one can split the game itself in two key segments: strategic part ("Should I buy the hotel?") and navigation part (spinning the dice and moving the figures through fields).

Transforming the board game into the multiplayer on-line environment raises issues with the least part. The main issue is the following – how to simulate the movement of the figures i.e. how to move the figures in parallel on multiple independent web clients. This problem raises more questions about the implementation of the game logic itself because the navigation is the essential part of the game. Should the game logic and the navigation be centralized in one place on a server, or distributed for different clients and then synchronized? The answer of this question really depends on the technology that we will be using.

According to the limitation not to use any additional software installed on the user's computer, we are certain that we will have to implement the navigation using the known web technologies.

III. ANALYSIS OF AVAILABLE TECHNOLOGIES

In this section we will present the available technologies and discuss which will be more relevant for our Cloud Computing SaaS solution. We will shortly discuss polling, long polling (comet programming) and push technology.

A. *Polling*

The solution based on polling is the standard solution that will probably come as an initial idea for the solution after the first look up on the problem. This would be the classical solution approach for the problem using the principles on which the web was originally defined. The solution would be implemented in the way that each of the users (clients) continuously sends request to the server asking for new data and gets responses for each request.

The problems that arise using this approach are obvious - the navigation definitely wouldn't be real time and the load the server would be enormous. Clients should constantly send requests in order to get new data from the server for each single event. For example, one event is the movement of the pawns on the map, the other event would be the value of the rolled dice, and for each event in the game the client would bomb the server with new requests. Too much requests mean increased overhead on the communication connection and will instantly reflect on overload of the server. A real drawback of performances will happen even with just a few parallel players.

We had goal not to limit the number of players and trying to find another solution that will reduce the problem of huge number of requests from the users. The analysis showed that a relevant technology that will solve this problem is based on push notifications where the server will notify users for new data from certain event, for example, when a player's pawn is moved or when the dice are thrown.

B. Long-Polling/Comet programming

This methodology is based on the fact that the client sends a request for new data to the server, and if the server currently doesn't have the requested data it doesn't terminate the request but keeps it open. At the moment when the server acquires the needed data it returns a response to the client and terminates the request. Immediately after completion of one request/response cycle like that, the client sends new request starting a new cycle. [2]

The solution implemented using this approach will significantly improve the performance compared to the solution implemented using a polling methodology. All this improvement relates to the cut of the number of request and response cycles. For example, if for the event "result of the rolled dice" we are using a polling methodology then a lot of empty request/response cycles will be performed needlessly over the period when one player is considering what to do when he gets his turn - "whether to buy a house or pay a toll."

Although this methodology cuts the number of empty request/response cycles still the approach remains the same. The pawn navigation among multiple users will be far better but still that will not be real time navigation. The usage of this methodology will partly relieve the server yet a single client will still send multiple requests asking for data needed for multiple events. This still represents a huge overhead of work for the server.

C. Web Sockets

WebSocket is a concept developed as a part of the HTML 5 initiative. This concept brings a new way of communication between clients and servers in the way that it simplifies the complexity around bi-directional web communication and with that provides much better performance than the other known technologies and techniques. This technology defines full duplex channel communication over a single TCP socket that makes WebSocket a new kind of PUSH technology. The architecture of usage in the cloud concept of the Monopoly game is presented in Fig.1 [3].

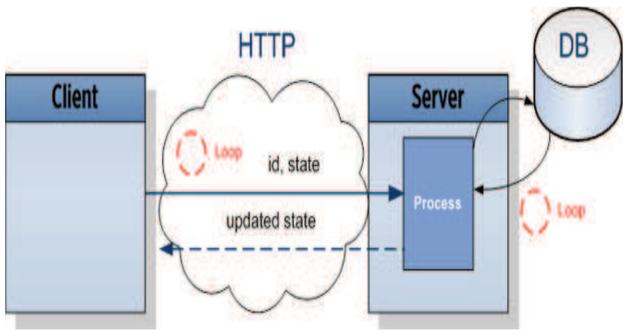

Fig. 1. Architecture of cloud solution using web sockets.

The whole specification of this technology is described in two parts, the WebSockets application programming interface standardized by W3C, and the WebSockets protocol that is standardized by IETF. The model of WebSockets communication is presented in Fig.2.

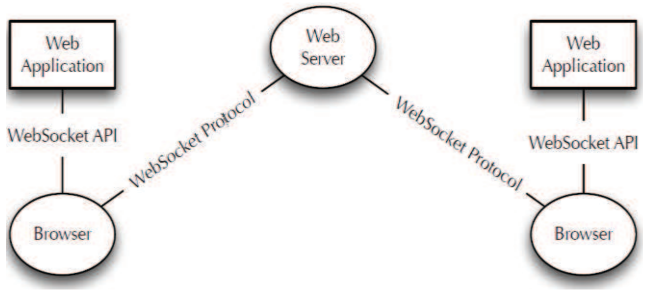

Fig. 2. Communication with web sockets.

The API defines an interface for communication between the browser and the web application, and each browser that supports WebSockets must provide and support this API to JavaScript web applications. The interface defines functions to that open, close WebSockets connection and functions to send WebSocket message.

The protocol describes the actual WebSocket communication and it is used by the API functions. When the application wants to send WebSocket message to a given end point, the application calls the API function to make the WebSocket connection. This function using the WebSocket protocol sends a handshake request to the given end point server, which is an HTTP request with an extra header fields. Using these extra fields the server calculates a hash value and gives the response to the client, by which it indicates if it supports WebSockets communication or not. If the handshake is successful, the client can send messages to the server, and the server can send messages to the client without being initiated from the client. [1].

The solution based on this technology will greatly releive the server from the request/response cycles. That means that the clients can be notified when a pawn is moved on the track or when the dice is roled by the server without asking constantly sending requests to the server asking "Is the dice rolled?".

IV. Selection of Web Technology

In previous section we have analysed possible web technologies to be used for real-time navigation as a basis for the Cloud Computing SaaS solution of Monopoly game.

Table 1 presents an overview of possible features for successful solution, the suggested technologies, and which technology supports certain feature.

Table 1: Comparison of different technologies.

Technology	Data update	Push update	Channels
Polling	X	/	/
Long-Polling	X	X	/
WebSockets	X	X	X

Comparing the possible implementations as technology to realize the navigation part in the Monopoly game, the implementation with Web Sockets stands as a definitive winner before polling and long-polling (comet programming) methodologies. The push possibilities of this technology makes the solution architecture much simpler, while getting much better performance i.e. real time navigation compared with implementations using the other two methodologies. Besides the better performance about the navigation, it significantly lowers the hardware requirements of the server side because it reduces a large part of the additional work that is generated through request/response cycles. Implementation of Web Socket server used in our solution enables channelled connectivity while separating all the events on which the navigation part should respond and in some way creating architecture that is fully event compatible.

When you implement a solution for the application in which the WebSocket concept of sharing data will be used, you must implement and WebSocket server that will provide the communication. There are both Commercial and Non-commercial implementations of WebSocket servers.

A. Non-commercial implementations

This type of implementation of the WebSocket server typically includes predefined software to be installed on the server, so that means that you should be supplied with servers in order to establish this environment. The fact that these implementations are non-commercial leads to the fact that they are a volunteer projects or student projects. For the same reason often happens that these implementations do not have well-written documentation, no support and in many cases there are limitations of languages and programming environments in which they can be integrated/implemented. One of the most popular non-commercial implementations that we analysed during the process of creation of the navigation part in the Monopoly game are: Socket.io [4], jWebSocket [5] and Xsockets [6].

B. Commercial implementations

The commercial type of the server implementations has better documentations and support than the non-commercial ones. Besides that, the commercial implementations almost always have some additional useful features to offer implemented in a wrapper API's over the standard WebSockets API. These features are related to the security, consistency, portability etc. We've done experiments using some of the most popular commercial implementations like Pusher [7] and Kaazing [8]. The last one requires the user to have his own server on which he can install the Kaazing server. That is not the case with Pusher and that is the main reason why we decided to Pusher instead of Kaazing.

V. SOLUTION

Led by the results from the previous analysis of the technologies and methodologies we came with the conclusion that it will be the best approach if the clients can get the required data using some push style methodology. In that way we will reduce the generated Internet traffic from the game. Besides the Internet traffic, by cutting the empty generated request-response cycles we reduce a lot of overhead work that the game server should do and with that we reduce the risk of stall moments on the server and reduce the server's hardware requirements. Most important by choosing the pushing methodology we gain a speed performance in the navigation.

In order to completely reduce the request-response cycles, decrease the server load and gain a little more in performance speed we decided to use push technology based on web sockets. The idea is to use the push methodology to update the clients with newest game data and with the goal to make the navigation real time. The following conclusions were also made.

If we make all navigation and major events that are non-vulnerable to run on the client side while the remaining logic stays on server side we solve much about the issue of performance. That way we will achieve that each client will be responsible for layout i.e. the presentation of the current situation on the game board and navigation of the pawns of the players will be processed and executed locally after receiving a push order from the server. Moreover, to simplify the work of the game server and make better architecture it will be better if we separate the game server and the WebSockets server. In that way the game server will be just another client in this communication process, and the web socket server will play a role of message router.

In order to enable the WebSockets communication we had to implement a WebSocket server. We than had choice to use complete WebSockets implementation or implement our own from scratch. There are several successful WebSocket server implementations on the market both open source and commercial. The main difference is that by using an open source implementation we will have to use an additional server and take care of its security and maintenance. That's why we decided to use a commercial WebSockets server implementation Pusher. It provides his API wrapper of the original WebSockets API, which provides additional functions, call-back methods, security mechanisms and etc. One of the additional features that Pusher provides is that allows multiple channels to be created between the clients targeting only certain groups of clients subscribed to these groups. Additional results of our analysis and experiments about the WebSocket server implementations can be found in the next section.

The solution architecture is presented in Fig.3 with both web socket and game servers.

The game runs as a process on a dedicated centralized game server. This process communicates and orchestrates the other web services, which are part of the game. Each client that wants to play a game will have to log in first (login is just another game service). Once the player is logged in, the main application takes care of further actions and the player receives the presentation layer from the server, the board, the panels, etc.

Besides the presentation layer, the application will open a WebSocket connection between the client and the WebSocket server. The game process also opens a connection between the process and the WebSocket Server, with the game process playing a role of another client. Since Pusher provides multiple channelled conversations over the same clients, each game event has its own WebSocket channel.

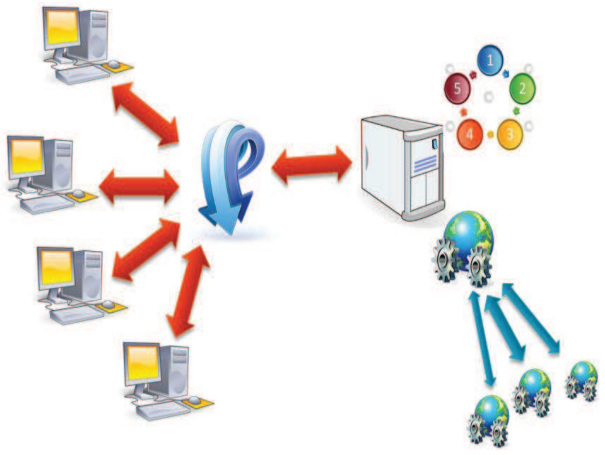

Fig. 3. Cloud concept using web servers and clients.

When all of the users are logged in and ready to play the game cycle starts running inner game logic. Each event that happen in the game server, asks the desired client or clients to perform some action giving them some order by appropriate Web Socket message. When the requested client makes an action, it sends the action result to the game server that continues with the game flow and then gives another order to client or clients when another event is expected to happen.

An example of the game flow is described in the following sequence procedure:
1) Game server: *Send message* to player1 to roll the dice.
2) Player1: *Roll the dice*
3) Player1: *Send message* to server "P1 rolled 2".
4) Server: *Send message* with info to all players "P1 rolled 2".
5) All Players: *Update info board* with the given message.
6) Server: *Send message* about movement to all players "Move P1 one field".
7) All players: *Move* P1 one field – The animation lasts half second
8) Server: *Send message* about movement to all players "Move P1 one field".
9) All players: *Move* P1 one field
The procedure continues until the game finishes.

VI. CONCLUSION

The concept of WebSockets as a new concept introduces a complete new way of communication and technology among web applications. This new concept provides a possibility for direct bi-directional asynchronous communication and offers increased performance by decreasing the server overload and communication. Therefore it makes this push technology to stand out from the rest and the WebSockets concept to be the new revolutionizing step as solution for web communication technologies.

In this article we have presented a solution how this technology was successfully used. We created a Cloud Computing SaaS solution of the Monopoly game and solved the challenge to realize real time multiplayer game navigation and on-line presence. The performance related issue was solved as a result of the performance that this technology offers for push messaging and the simple architecture of the solutions.

The decision to realize the navigation to be processed locally and guided from a centralized source represents and additional but big detail in the complete picture.

Ours is just one example of the successfully completed multiplayer cloud game by the means of WebSockets. This technology opens a wide range of new applications to be developed, more complex cloud games even completely cloud 3D games. There is a great probability that WebSocket will replace the other known technologies like polling, long-polling i.e. comet programming especially in the applications when the on-line presence of the user are live consistent data are playing crucial part.

REFERENCES

[1] Marco Casario, Peter Elst, Charles Brown, Nathalie Wormser and Cyril Hanquez, "HTML5 WebSocket," *HTML5 SOLUTIONS: ESSENTIAL TECHNIQUES FOR HTML5 DEVELOPERS*

[2] Harri Hämäläinen, "HTML5: WebSockets," *Aalto University, Department of Media Technology*

[3] Dmitry Sheiko "WebSockets vs Server-Sent Events vs Long-polling" (on-line) http://dsheiko.com/weblog/websockets-vs-sse-vs-long-polling

[4] *Socket.IO the cross browser WebSocket for real-time applications* (on-line) http://socket.io/

[5] *JWebSocket Open Source Java WebSocket solution* (on-line) http://jwebsocket.org/

[6] *Xsockets Real-time WebSocket solution* (on-line) http://xsockets.net/

[7] *Pusher real-time messaging system* (on-line) http://pusher.com/

[8] *Kaazing WebSocket technology* (on-line) http://kaazing.com/