A Web-based Version of a Trivial Game to Promote Galician Culture *

Miguel R. Luaces, Oscar Pedreira Ángeles S. Places, and Diego Seco

Databases Laboratory. University of A Coruña. 15071 A Coruña, Spain
{luaces,opedreira,asplaces,dseco}@udc.es

Abstract. We present in this paper the architecture and some implementation details of a web-based version of a Trivial game. Our implementation achieves such a high degree of interactivity between the players that they perceive the game as being played in real-time. More importantly, no plug-in or applet is used in the architecture of the system. These properties are achieved by means of a carefully designed architecture that uses AJAX (*Asynchronous JavaScript and XML*) for data exchange. Using this approach, it is possible to develop any type of web-based collaborative software with few load on the web server. In the paper, we analyze traditional architectures for web-based applications and we show how our approach overcomes their limitations. Furthermore, we proof the efficiency of our approach by means of an empirical comparison.

Key words: web application, user interface, AJAX, collaborative software, e-learning

1 Introduction

The maturity of Internet users and the quality of connections and services available are increasing the demand of interactivity in web applications, not only between the user and the system, but also between the users themselves. However, the characteristics of traditional web applications prevent developers from building collaborative applications or games that require real-time interaction between the users [1]. This is due to two main reasons:

- *Clients cannot exchange information.* Connections in web applications are always established between a client and the server, but never between two clients. Hence, all data exchange must be done through the server.
- *A web server cannot start data transfers.* Web servers can never communicate the information received from one client to the others unless the clients explicitly request it, thus restricting the interaction possibilities between the clients.

* This work has been partially supported by "Ministerio de Educación y Ciencia" (PGE y FEDER) ref. TIN2006-16071-C03-03, by "Agencia Española de Cooperación Internacional (AECI)" ref. A/8065/07, and by "Xunta de Galicia" ref. 2006/4 and ref. 08SIN009CT.

As a consequence, applications where users collaborate or interact which each other in real-time to perform a task have to be implemented using *plug-ins* or a similar type of software for the web browser that controls the exchange of messages between the users. The only alternative without this type of software is that each client requests frequent and periodic updates from the server to retrieve the data that has changed in any other client. However, if data change frequently in the clients or the change has to be perceived as real-time, the load in the web server will be very high because many new pages will have to be created and sent constantly. This results in a limitation in the maximum number of users that can interact. Nevertheless, this approach is better than the previous one in the sense that users do not have to install any plug-in, which is an insuperable restriction in application domains where users do not have the required expertise level.

We have developed a software architecture specifically designed to simulate interactivity between users of a collaborative web application. Users perceive their interaction as real-time, just like if they had a direct connection between them. This architecture has been used to implement a virtual version of the classic board game *Trivial Pursuit* with two important advantages over other applications of this type: it does not require players to download and install any software for the web browser, and a large number of games with many players in each can be played simultaneously in an ordinary web server.

The rest of the paper is organized as follows. In Sect. 2 the rules of *Trivial.gz* are described in order to show the level of interactivity that can be reached with this new approach of web application development. Then, in Sect. 3, we present the differences between the architecture of traditional web applications and the architecture we propose, and we describe AJAX in more detail, showing its advantages for the systematic development of interactive web applications. After that, we present a detailed description of the application architecture in Sect. 4. This development allowed us to evaluate and compare our approach with respect to traditional approaches to web application development, which is presented in in Sect. 5. Finally, Sect. 6 presents our conclusions and some ideas for future work.

2 Trivial.gz

Trivial.gz is an initiative of the Galician Socio-Pedagogic Association (AS-PG, from the Galician name *Asociación Socio-Pedagóxica Galega* [2]) to increase the usage of Galician language on the Internet and among the young people, and it was sponsored by the Galician government. The game was presented during the computer party *XuventudeGaliza.Net*, which took place in Santiago de Compostela in April 2006. The game can currently be played at the web server of the Galician Socio-Pedagogic Association (`http://www.as-pg.com/trivial.gz`). Figure 1 shows a screenshot of a game being played.

The main objective when the development started was to create a web-based game similar to the Trivial Pursuit board game which could be played on any

Fig. 1. Screenshot of the game

web browser without having to download any plug-in or applet. Our version of the game has some differences with respect to the original board game in order to get the most out of the virtual environment and to minimize the effects caused by the players not sharing the same physical space during the game. Moreover, a big effort was devoted to simulate in a web page the interaction between the players and the actions like rolling the die, moving the tokens, watching the positions and movements of the other players, etc.

In a rough description, the goal of *Trivial.gz* is answering correctly questions of three different difficulty levels and six different subjects: *Culture and Show Business*, *Geography*, *History*, *Language and Literature*, *Science* and *Our World*. All questions have three possible answers of which only one is correct. The board is an hexagon divided into squares of different colors, each representing one of the subjects of the questions. When the game starts, all players start from the central square. The player who has the turn throws the die (by clicking on an animation of the die rolling) and moves on the board in any direction as many squares as the number in the die. When a square is chosen, the same question is showed to all players who can try to answer the question before the time ends. If the question is answered right points are awarded to the player independently of whether the player had the turn or not. If the player with the turn fails the question, the turn passes to the next player. To win the match, the player first has to collect the six *wedges* that are awarded when the question on a vertex of the board is answered right. Then, the player has to proceed to the center square and answer correctly a question from a random subject.

A detailed description of the game rules is outside the scope of this paper. However, we think it is interesting to discuss the modifications that were done to the original rules of the game in order to improve the experience of the players:

- *The game board is updated in real time.* A player can see all the tokens, the die value, who has the turn, and the points of each player. This creates the sensation that the players share the same virtual space.
- *Players can talk.* A chat was added to the game page so that players can communicate. It simulates the verbal interaction between players.
- *Wedges can be lost.* If a player fails the question in one of the squares where wedges are awarded, the player looses the wedge. This makes the game more dynamic and enables players that are losing to recover.
- *It is easier to win wedges.* There is a special square in each side of the board that moves the token to one of the special squares. This makes games faster.
- *Everybody plays.* When the player with the turn chooses a square, the question is presented to all players. Everybody can try to answer the question and points are awarded to everybody that gives the right answer. Furthermore, it can be seen who has already answered the question right or wrong in order to increase the perception of playing with more people.
- *Limited time.* There is a time limit of 30 seconds to answer for the player that has the turn. The time for all the other players is limited to the time required by the player with the turn. This avoids long waits.
- *Player history.* The server keeps track of all the games played, the points achieved by each player, and the statistic of questions answered right for each subject. This gives the game an additional dimension because players can compete not only to win one game, but also to be the one with more games won, with more points, or with better statistics.
- *Solitaire game.* The game can be played by a single player.

The usage statistics of the game are very encouraging. It currently has more than 4400 registered users and more than 6000 questions. The average number of visits is around 50 visits a day. Furthermore, new functionalities are being developed to increase these statistics. The Trivial.gz championships are an example of these functionalities. An administrative tool to configure the championships (i.e. subjects, number of matches that each contestant can play, difficulty, etc.) has been developed and the game engine has been extended. Four championships have been carried out since May 2008 (when this functionality was released) and the average number of contestants is around 36.

3 Differences with Traditional Web Applications

The architecture of traditional web applications follows one of these two philosophies:

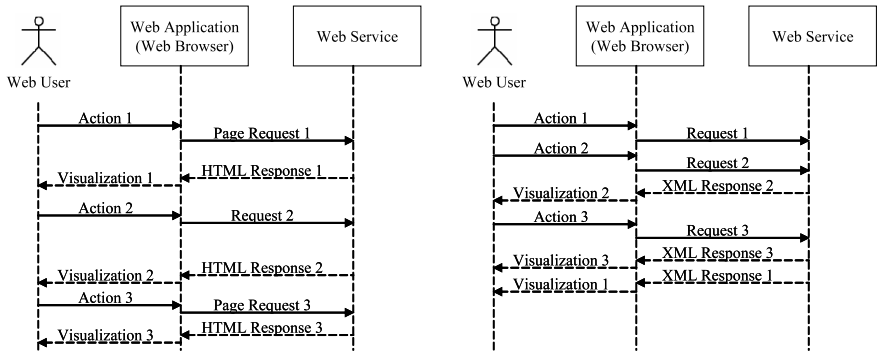
- *Server-side applications.* All processing is performed on the web server. Each client request implies a processing time in the server and sending a complete web page to the client. This architecture is not very scalable because the number of pages that the server has to process and send to the clients grows with the interactivity of the application and the number of simultaneous clients.

– *Client-side applications.* In this type of architecture as much processing as possible is performed in the client, thus minimizing the information exchange with the server. This type of applications are usually implemented by means of web browser plug-ins that have to be downloaded, installed and configured. Another approaches use Java applets that require the Java Virtual Machine to be installed and configured. In any case, this philosophy requires some level of expertise from the users, which limits its general use.

An intermediate approach uses scripts in the web pages so that the client-side of the application has a certain amount of processing capabilities without having to install a plug-in or the Java Virtual Machine. AJAX (*Asynchronous JavaScript and XML* [3]) is the name of a new philosophy in the field of web application development. In fact, AJAX is not a new technology but rather a combination of a number of already existing different technologies. The central element is the asynchronous usage of the *XMLHttpRequest* API present in all web browsers of the current generation. This allows a web page that is being visualized at the client-side to use a script language function to request some information from a web server without blocking the user activity. The web server returns the information requested using short XML [4] messages and the web browser invokes a specific script function that can process the response and modify the web page accordingly. Google has been a pioneer in the use of AJAX, as can be seen in Google Suggest, Google Maps or GMail [5].

Figure 2 shows a sequence diagram represented with UML that describes this behaviour. In both figures a user invokes three actions in the user interface of the web application. Figure 2(a) shows the behaviour in the case of traditional web applications. In this case, the user has to wait until the action ends before invoking the following one. Furthermore, the processing time in the server and the amount of information exchanged between the client and the server is usually quite high. Figure 2(b) shows the behaviour in the case of using AJAX. In this case users perceive a higher response speed. They do not have to wait for an action to end before invoking another action because the data exchange is performed asynchronously and long operations do not block the user interface. Moreover, in traditional web applications each content update requires a complete reload of the web page, whereas in a web application using AJAX the information in the XML message is used to redraw the appropriate section of the user interface. Additionally, the processing time in the server and the amount of information exchanged is smaller because the server does not have to create and trasfer complete web pages, but only short XML messages. Furthermore, given that the processing time in the server and the amount of information exchanged between the server and the clients is reduced, AJAX-based web applications can include more interactivity than traditional applications because the remaining processing time and bandwidth can be used to handle a higher number of simultaneous requests.

A number of development tools have appeared around AJAX to make its usage more easy. One of them is *Direct Web Remoting (DWR)* [6]. This open source library has two advantages over the direct use of AJAX. First, it enables



(a) Traditional web application (b) AJAX web application

Fig. 2. Client-server interaction in different application models

the JavaScript code in the client-side to use transparently Java classes in the server-side. That is, it enables the developer to use AJAX in a similar way to CORBA or RPC. This is achieved by dynamically generating JavaScript code that encapsulates AJAX-based calls to the Java classes in the server. The second advantage is that DWR provides the developer with a number of tools to make easier the update of the web page contents in the client-side.

These two technologies are the center of the architecture of our application. However, they do not solve the problem of creating web applications that allow users to interact in real time. Even though it speeds up data exchange between the server and the clients, it does not change the architecture of web applications. That is, data exchange is still performed between the clients and the server and never between clients, and the server still cannot take the initiative of sending the data it has just received from a client to the other ones.

In order to achieve a high degree of interactivity between the users, in our architecture each client issues frecuent periodical requests to the server and receives the relevant information regarding the state of the game and the other players, which is used to change the user interface accordingly. To keep track of the current state of the game, the server implements a state-machine that is controlled with the information sent by the player that has the turn. Requests to the server can be issued more in response to some actions of the player (such as throwing the dice or answering a question).

This approach cannot be implemented with the traditional architecture of web applications because the amount of information that had to be exchanged between the server and the clients is too high. In order to minimize the processing time in the server and the traffic between the clients and the server, we use AJAX for the communication between the clients and the server. By exchanging information by means of short XML messages, AJAX frees the web server from the creation of complete web pages when the clients requests arrive. Furthermore, the logic of the application is split between the client and the server in such a

way that the processing time in the server and the traffic between the server and the clients is minimized. Finally, the functionality in the client-side can be implemented using only JavaScript, thus no special software has to be downloaded or installed.

4 Detailed System Architecture

Figure 3 presents a general view on the architecture of the application. Just like in any web application, one can find two different parts: the server-side module of the application that runs in a web server and it is implemented using *Java Server Pages (JSP)*, and the client-side module that runs in the web browser of the client and it is implemented using JavaScript and dynamic HTML. There is a part of the application that deals with functionality such as user registration, configuration of lists of friends, or querying and browsing statistics of the players, which is implemented as an ordinary web application whose description is out of the scope of this paper. Instead, we will focus on the description of the game control and the simulation of real-time interactivity.

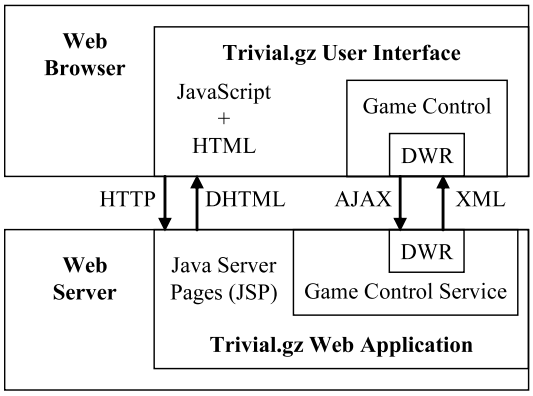


Fig. 3. Application architecture

There are two different types of players in the game: the one that currently has the turn, and all the others. The first one has the control of the game and generates events that produce the update of the game state (e.g., rolling the die, choosing the square, or answering the question). The other players only generate an event when they answer the question. The server controls each game being played by means of a state-machine. During the game, the server goes through the states according to the actions invoked by the player that has the turn. The other players do not have any effect on the state-machine, they just query the server periodically to retrieve the current state of the game in order to update their user interface.

The states in the state-machine of the server are the following:

- *Initial state.* Just before the player that has the turn rolls the die. There is no information to be sent to the other players.
- *Dice thrown.* The player with the turn rolls the die and gets a number. The other players receive this number when they request the game state in order to update their user interface.
- *Square chosen.* The player with the turn chooses a square to move and receives a query for that square that is selected randomly by the server. The rest of the players receive the square selected and the question when they request the game state.
- *Answering a question.* Whenever a player without the turn answers a question, the server updates the points of that player and informs the other players of the result. Therefore, all the other players can now whether one player has answered the question right or wrong. When the player with the turn answers the question, the time for answering ends. If the player gave a wrong answer, the turn passes to the following player.

Internally, the server-side module keeps a list of the games that are being played at any given time. Each game consists of a list of players and a list of questions ready to be sent to the players. For each player, the server keeps in memory the points achieved, the wedges won, and the statistic of questions answered right for each subject. The list of questions acts like a memory cache to reduce the frequency of database accesses. Hence, instead of issuing a query to the database each time a question is needed, there is only one database access to retrieve a set of questions that are used during the game. Only when all the questions are used a new database access is performed.

The client-side modules consists of a Dynamic HTML page with JavaScript code. Its operation is based on a JavaScript timer that requests the game state every four seconds and updates the user interface accordingly. When the user invokes actions on the user interface, the JavaScript code informs the server of the event and modifies the user interface with the information retrieved. The time between updates can be easily configured. A longer time reduces the real-time perception of the game but allows for a longer number of simultaneous games in the server. On the other hand, a shorter time improves the real-time perception but requires more computing power on the server side. The time that is currently being used has been empirically chosen to achieve a real-time perception of the game while using an average computer as the web server.

Figure 4 shows the development of a game turn. The numbers in the figure match the following enumeration and represent a temporal ordering of the event sequence. First, the JavaScript timer determines the moment to request information from the server (1). After that, a JavaScript function in the client-side module that requests the current game state from the server is invoked (2). The server receives the request and delegates its fulfillment in the business model (3). The business model implements the state-machine that controls the game and computes the information to be answered according to the current state and the current event (4). Then, the server response is encapsulated and

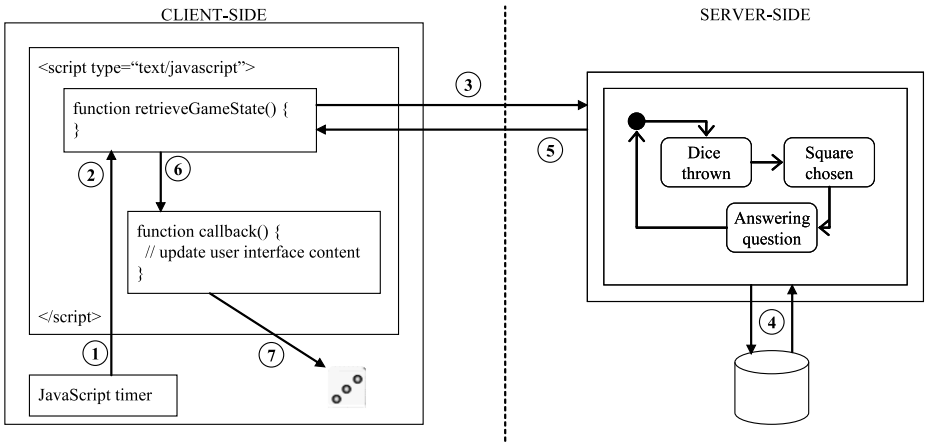


Fig. 4. Communication protocol

sent to the client. An object is built with the information retrieved from the business model and sent to the client (5). The client receives the response, extracts the information from the object, and uses the information to update the user interface content (6). Finally, a new turn starts by showing the animated picture that represents the die (7).

The state-machine that controls the game solves one of the most important drawbacks in applications where clients perform some processing: *to make sure that the messages received in the server are according with the state of the application.* This drawback is especially important in web applications where client-side processing is implemented using JavaScript code. Several applications (e.g. Firebug [7]) can be used to analyze the script code in web applications. Moreover, messages sent to the server can be modified using applications or plug-ins such as Tamper Data [8]. Thus, a user with the required expertise level could cheat easily. For example, a user with the turn could send a message with his favourite *chosen square* without sending the message *throw the dice* if the state-machine had not checked the state of the matches. Therefore, the state-machine must verify all the messages and parameters received by the server.

5 Experimental Evaluation

The *Trivial.gz* is installed in a 3.2 Ghz Pentium IV with one gigabyte RAM. During its presentation at the *XuventudeGaliza.Net* party, the web server received almost a million *hits* (web server requests) and more than 600 visits (defined as a set of consecutive hits originating from the same computer). Most of the requests were concentrated in the hour when a *Trivial.gz* competition took place. More than 200 players competed to achieve more points than the rest. During this hour, more than 800 games were played with an average of

three players per game. Exactly 776 games were played completely. The games that did not end because all the players left the board were not registered.

Table 1 shows some figures regarding the amount of information transferred by the web server executing *Trivial.gz* and a comparison to the amount of information transferred if the application were built using a traditional web architecture.

	Traditional Application	Trivial.gz
1 message, 1 game, 1 player	35 KBytes	0.31 KBytes
1 game, 1 minute, 3 players	1575 KBytes	14.1 KBytes
100 games, 10 minutes, 3 players	1538.09 MBytes	13.77 MBytes
Network traffic during these 10 minutes	2.56 MBytes/s	23.50 KBytes/s

Table 1. Empirical data

The HTML code of the web page occupies 35 kilobytes without taking into account neither the images nor the JavaScript code that sends the messages to the server using DWR. On the other hand, the DWR object that the web server sends to each client occupies 313 bytes, i.e. 0.31 kilobytes. As it was explained in Sect. 4, the game information must be updated by each client frequently in order to achieve a real time perception of the game. If we consider an update time of 4 seconds, just like it is currently configured in the application, a traditional web application would require the web server to send the 35 KB of the web page 15 times per minute. This means a total of 525 KB per minute and player, whereas with *Trivial.gz* this amount of information is reduced to 4.7 KB per minute and player.

We can suppose that 100 games with an average of three players were played during 10 minutes in the party where the *Trivial.gz* was presented. In fact, more than 800 games were played in an hour, so our supposition is quite conservative. Considering this activity, 1.5 Gigabytes of information would be transferred through the network in a traditional web application, whereas only 14 Megabytes were sent in *Trivial.gz*. This represents that a traditional web application requires 107 times more space for the same information. As a consequence of this amount of information, a traditional web application would require a bandwidth of 2.56 MBytes/s in the best case (assuming that the traffic distribution is uniform). However, *Trivial.gz* only needed a bandwidth of 23.50 KBytes/s.

6 Conclusions and Future Work

We have presented in this paper the architecture of a web application that implements a trivial-like game where players can follow the game in real time. This was achieved without using any plug-in or applet, which means that the game can be played in almost any computer without the common problems associated to the instalation of these components.

We have described how the AJAX philosophy is used in the implementation for the data exchange between the server and the clients. Moreover, the message exchange protocol designed for *Trivial.gz* and a general architecture for collaborative web applications are also described. These three improvements over traditional web applications minimize the amount of information that has to be transferred between the clients and the server. We believe that this approach can be used to develop any type of collaborative software using a web application with a small load on the web server.

Finally, we have performed an empirical comparison between *Trivial.gz* and a web application based on a traditional web application architecture. This comparison shows the advantages of using our architecture in terms of network bandwith and processing capabilities of the web server.

The application has been a success. The game has a solid player base and it is being used to promote the use of the Galician language among young people. New questions are submitted by player regularly and a board version of the game is being developed. As lines of future work, we are currently working on the improvement of *Trivial.gz* to allow games with a larger number of players. Another improvement that is currently being implemented is the adaptation of the application to make it completely configurable by the final users. When this is achieved, any user will be able to install the application, decide the subjects, and define the questions. A possible field where this application can be used is in education with teachers configuring *Trivial.gz* games with the questions of the lessons they teach in the classroom.

A different line of future work is the study of the requirements of other types of collaborative web applications, such as distance education or collaborative work, and the adaptations that the architecture requires. In this sense, a web-based version of the popular game *Scrabble* [9] is being developed with the same goals in mind.

References

1. Paulson, L.D.: Building rich web applications with ajax. IEEE Computer **38**(10) (2005) 14–17
2. Galician Socio-Pedagogic Association: Web Site. Retrieved from `http://www.as-pg.com/` in October 2007 (2007)
3. Garrett, J.J.: Ajax: A New Approach to Web Applications. Retrieved from `http://www.adaptivepath.com/publications/essays/archives/000385.php` in October 2007 (2005)
4. World Wide Web Consortium: Extensible Markup Language (XML). Retrieved from `http://www.w3.org/XML/` in October 2007 (2006)
5. Google: Google Tools Web Site. Retrieved from `http://www.google.com/intl/en/options/` in October 2007 (2007)
6. Direct Web Remoting (DWR): Web Site. Retrieved from `http://getahead.org/dwr` in October 2007 (2007)
7. Firebug: Web Site. Retrieved November 2007 from `http://getfirebug.com` (2007)

8. Tamper Data: Web Site. Retrieved November 2007 from `http://tamperdata.mozdev.org/` (2007)
9. de Bernardo, G., Cerdeira-Pena, A., Pedreira, O., Places, A.S., Seco, D.: Scrabble.gz: A web-based collaborative game to promote the galician language [to appear], IEEE Computer Society Press (2008)