2009 Second International Conferences on Advances in Computer-Human Interactions

SCRABBLE.GZ: A web-based collaborative game to promote the Galician language [*]

Guillermo de Bernardo Ana Cerdeira-Pena Oscar Pedreira Ángeles S. Places Diego Seco
University of A Coruña
Database Laboratory
15071 A Coruña, Spain
{gdebernardo,acerdeira,opedreira,asplaces,dseco}@udc.es

Abstract

We present in this paper a web-based version of a Scrabble game, describing its architecture and some implementation details. This architecture makes possible a high degree of interactivity, so that the players perceive the game as being played in real-time. Furthermore, no client-side plug-in or applet is used. These properties are achieved by means of a carefully designed architecture that uses AJAX (Asynchronous JavaScript and XML) for data exchange. This architecture guarantees low load on the server, so complex computations relative to the game logic can be done in real-time. Moreover, data structures and algorithms were designed to efficiently access a custom Galician dictionary, which supports the game functionalities. We show in this paper how this data structures and algorithms provide an efficient method to create a Scrabble move generation algorithm. We also show how the combination of these with the architecture proposed provides a fully interactive web application that can handle complex calculations over a very large lexicon with real-time appearance.

1. Introduction

The growing importance of Internet and the World Wide Web has made it one of the most important forums to spread the culture and the language of a nation, to the point that the potential of a culture is known that is related with its presence in Internet. Young people are the most frequently users of the Internet, as well as being the future of any culture. Therefore, e-entertainment applications (i.e. games through the Internet) must be developed to guarantee the presence of a culture in the World Wide Web.

[*]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.

Furthermore, 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 [5]. 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.

- *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.

Therefore, applications where users collaborate or interact with 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 updates from the server to retrieve the data that has changed in any other client. However, if data changes frequently on the clients or the change has to be perceived as real-time, the load on the web server will be very high because many new pages will have to be created and sent constantly. This results in a limitation of 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.

New techniques and technologies have been emerging during the Web 2.0 era. They can help in the development of collaborative web applications. We have developed at the Databases Laboratory of the University of A Coruña a

978-0-7695-3529-6/09 $25.00 © 2009 IEEE
DOI 10.1109/ACHI.2009.12

software architecture specifically designed to simulate interactivity between users of a collaborative web application [4] using these new techniques. 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 install any software for the web browser, and a large number of games can be played simultaneously on an ordinary web server.

This paper presents a web-based version of the popular game Scrabble that has been developed using the same architecture. Furthermore, one of the main objectives in this work was the development of structures to efficiently store and manage a vocabulary for the game. These structures could be used in many other applications.

The rest of the paper is organized as follows. In Sect. 2 the rules of *Scrabble.gz* are described in order to show the level of interactivity that can be reached with this approach. Then, in Sect. 3, we present a detailed description of the application architecture and some implementation details. This development allowed us to evaluate and compare our approach with respect to traditional approaches to web application development, which is presented in Sect. 4. Finally, Sect. 5 presents our conclusions and some ideas for future work.

2. Scrabble.gz

Scrabble.gz is a web-based version of the Scrabble board game for the Galician language. Adaptation of classic games like Scrabble, with great qualities as a pedagogic tool to increase language capabilities, is expected to be a goal in promotion of Galician language. One of our objectives in the development of this web version was that the application could be played on any web browser without having to download any applet or plug-in. We also tried to simulate the normal user interaction in real board game. Another of our main purposes was to develop a complete, efficient dictionary for the Galician language that could store a vocabulary and support complex queries that satisfied the needs of this game and those other applications that could possibly use it.

The original Scrabble board game consists in placing words, made up using tiles, on a 15x15 board. Tiles represent a character or digraph and have an associated score. The goal of the game is to obtain the best score from one's words, which must be placed crossing another word previously played. In his turn, a player can pass, change some of his tiles or play a word. Match finishes when a player uses all the tiles in his rack and there are no more remaining, or when nobody can play a word.

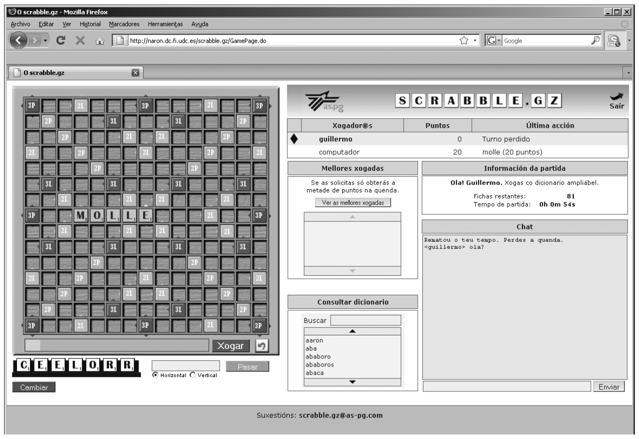
As well as these general rules, we will show also the most important differences between our web version of Scrabble and the original board game:

- Players have information about the state of the match, specifically score, number of tiles left, and last actions of all the participants. This helps the user focusing on the match.
- Players can talk during the match, using a chat window. This simulates the verbal interaction between players. The sensation of real-time interaction relies mostly on this point and the preceding one.
- The computer automatically validates words played, using a reference dictionary of Galician accepted terms. Current application provides two dictionaries. Players can access this dictionary during the match to check their words before playing them. The set of valid words is hence well known, so move refutation is not needed.
- Users can suggest new words to be added to the dictionary. Move refutation, as understood in the classic game, is replaced by this possibility. Currently, suggestions are only allowed for the complete version of the dictionary, as the smaller one is supposed to be a stable reduced version.
- Players can require suggestions from the computer. They will obtain the best moves for them, sorted solely by their total score. Returning a set of words helps the player to select the one that fits his own purposes. These suggestions imply a loss of points from the score of the turn to keep the match even. The real-time sensation relies here on the ability to find this set of moves in a short time.
- Users can play against the computer. The application will behave like a normal player in these matches, playing a good move in its turn. Efficiency in this calculation is needed to avoid long waits for the user while providing a good level machine player.
- The set of tiles and scores for each one are adapted to Galician. This adaptation was done by finding letters frequencies in a huge corpus, which was also the basis to create the main dictionary.
- Each player turn has a limited time. In the current application, it is set to 45 seconds. This value was empirically determined from the average time to play a word. Configuration of this parameter is very important to avoid long waits while giving the players time enough to move.
- The server keeps track of all the games played, storing information about matches won and points obtained

for all users. Therefore, players can compete to win one game, but also to be the one with most games won, most points, or the best statistics.

3. Implementation

3.1. Interface design



	Xogador@s	Puntos	Última acción
♦	guillermo	0	Turno perdido
	computador	20	molle (20 puntos)

Se as solicitas só obterás a metade de puntos na quenda.

Información da partida

Ola! Guillermo. Xogas co dicionario ampliábel.

Fichas restantes: **81**
Tempo de partida: **0h 0m 54s**

```
Rematou o teu tempo. Perdes a quenda.
<guillermo> ola?
```

aaron
aba
ababoro
ababoros
abaca

Suxestións: **scrabble.gz@as-pg.com**

Figure 1. Screenshot of the game

As we stated before, interactivity of the game was one of our main objectives in the development of the application. Hence, the design of the game page was done very carefully. All extra features to the original game are shown clearly to ensure a better comprehension of how the game works. In addition, the most important information about the actions of the other players is always being updated so that the user knows perfectly the state of the match. Figure 1 shows a screenshot of the current user interface for the game page.

The game page is divided in two main parts. On the one hand, elements needed for the normal game (i.e. the board or the player rack) are placed in the left hand of the screen. These elements are grouped together to make easier the interaction. The board state and the user rack are always updated, including highlighting of last tiles placed. On the other hand, all other functionalities are placed on the right, grouped to be accessed quickly. Information about the match and about the other players is also updated regularly. In order to help users to understand the enhancements done in our web-based version of the game, functionalities to request for suggestions or query the dictionary are placed in the central part of the screen, near to the board. The dictionary can be accessed by starting string, and looks like a list of words stored on the client. One more time, efficiency on the information exchange with the server allows a real-time interaction sensation. To provide the communication among players, a chat window is placed in the right part of the window. Communication is supposed to be done mostly out of one's turn, so updates of the chat window and other match information are done independently.

The layout of this interface is designed very carefully to help the player to focus in the match. Regular updates of the interface lead the users to feel like playing at home, around the same table, and chatting with the other players. Moreover, enhancements integrated in the user interface provide a more dynamic game, where players can obtain better solutions in a shorter time and some problems of the normal game (as move refutation) are carried off automatically.

3.2. System architecture

Architecture of traditional web applications follows one of these philosophies:

- Server-side applications. All processing is performed on the web server. Each client request implies creating and sending a complete web page to it. This architecture is not very scalable.
- Client-side applications. As much processing as possible is performed in the client. These applications are usually implemented by means of web browser plugins that have to be downloaded, or Java applets that require the Java Virtual Machine to be installed and configured. 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 has a certain amount of processing capabilities without having to install a plug-in or the Java Virtual Machine. AJAX (*Asynchronous JavaScript and XML*, see [2]) is a new philosophy based on this. It uses the *XMLHttpRequest* API present in all browsers nowadays. This API allows a web page to request information from the web server without blocking the user activity. The web server returns the information requested using short XML [6], which will be processed by some script function on the client to update the page state.

This approach has many advantages. First, 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 transfer complete web pages. Hence, in AJAX-based web applications the remaining processing time and bandwidth can be used to handle a higher number of simultaneous requests.

The use of AJAX has been increased a lot in the last years, when many web applications have been adapted to work using this philosophy so that their user interface becomes more user-friendly. Because of this, a high number of tools have appeared around AJAX to make its usage easier. One of these tools is *Direct Web Remoting (DWR)*, an open source library that simplifies the use of AJAX encapsulating its requests with its own script code.

Our application is based on this AJAX philosophy, using DWR to manage asynchronous requests. However, our game requires a high degree of interactivity, so we need to simulate information exchange between clients. The use of AJAX increases the efficiency of data exchange, but does not change the request-response model that is used in all web-based applications. To achieve this sensation of real-time interaction, clients periodically query the match state using AJAX requests, and update the user interface according to the information they receive about the match and the other players. Some client actions require extra information from the server, which is requested using the same method.

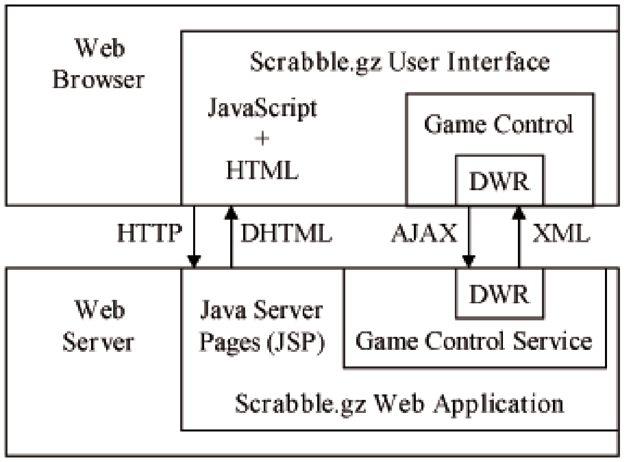


Figure 2. System architecture

Figure 2 presents a general view on the architecture of the application. Just like any web application, one can find two different parts: the server-side module and the client-side module. The first one is implemented using Java Server Pages (JSP), and the second one is implemented using JavaScript and Dynamic HTML. There is a part of the application that deals with functionality such as user registration or browsing statistics of the players, whose description is out of the scope of this paper. Instead, we will focus on the description of the game control and simulation of real-time interactivity.

The server-side module keeps the state of all matches being played and processes client requests according to this state. Submission of words, requests for suggestions, and all the requests related to a proper turn action are only allowed to the user in turn. Some other requests are allowed at any time. Wrong or unsynchronized requests are discarded. In this way, although most of the control is done on the client-side, the server-side can force some state changes to prevent a possible match lock, for instance, when a client falls down from a match. Server-side control of the matches was developed using a state machine that determines the valid actions for the player in turn and other information about the match.

The client-side module consists of a Dynamic HTML page with JavaScript code. Its operation is based on a JavaScript timer that requests the game state every few seconds and updates the user interface. When a user invokes actions on the user interface, the script code informs the server of the event and modifies the user interface with the information retrieved. This information depends on both the state of the match and the invocated action. As information is periodically synchronized, state control on the client-side works perfectly most of the time. Unsynchronized state changes that might lead to redundant or unsynchronized requests to the server will be quite unusual and will be corrected by the server-side. The time between updates has been empirically chosen to achieve a real-time perception of the game while keeping the low processing load on the server side.

3.3. Data structures and algorithms

Scrabble.gz, like other language-based games, requires an implementation of data structures and algorithms that grant a fast and flexible access to a big list of words. The efficiency of this access will allow our application to do the complex calculations needed in short time. This issue, along with the proposed system architecture, supports the real-time needs that are the main goal of the application. We will discuss in this section some details about these elements.

The data structure used for the dictionary is a variant of a GADDAG directed acyclic graph shown in [3]. This is a minimized graph where words are stored in its paths. Each edge of the graph has an associated character, so a path to a final node represents a valid word. Minimization of equivalent states leads to a small structure which can be accessed easily as a normal word *trie*. The main difference between this graph and a normal word graph is that the vocabulary to be accepted is different from the one inserted in the dictionary. We can see in the Figure 3 how for each word that should be accepted by the dictionary, the structure holds all of its rotations in a special way. The purpose of doing this is being able to start searching in any position of any word, thus being able to do complex queries efficiently. This method of construction increases the size of the resulting structure, but we will see in further sections that spatial efficiency of this approach is really high, what makes this problem less important.

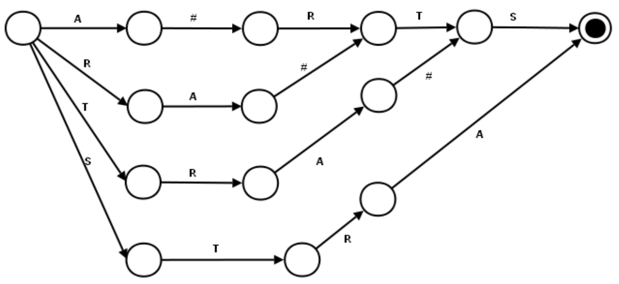


Figure 3. Graph structure. Variations for word "arts"

Our implementation is based on this directed acyclic word graph, stored in an array structure, but its design is adapted to ensure a high capacity while keeping the limited search complexity. Figure 4 shows this storage structure. Nodes and their leaving edges are stored consecutively, and edges for each node are lexicographically ordered. Each node and edge is stored in 32 bits, so the size of the array is $O(|E| + |N|)$, where E represents the edges set and N the nodes set. Each node contains a bitmap of characters where bits set to 1 indicate an output edge for the character corresponding to this position. Furthermore, each edge contains the destination node's position in the array and a mark to indicate if the path to that node represents a valid word. Paths to valid words can be marked using just 1 bit, so 31 bits can be used for the array indexing. Thus, our implementation will be able to hold a huge, potentially unlimited, number of words. This is because we minimize on the fly our word graph during construction. Regularity of the language ensures that the growth of the structure will be sub linear in the number of words added.

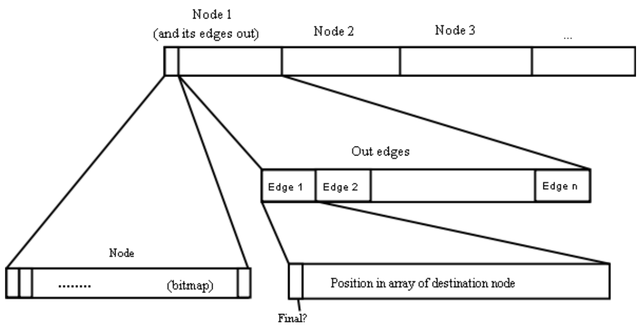


Figure 4. Graph storage

As we noted, each node contains a bitmap where all possible characters must be coded. This means a limited number of characters can be introduced in our dictionary. However, this limitation has not been considered a problem for us because the number of different characters in our Galician dictionary is assured to be little enough. Hence, the only limitation to our dictionary is the amount of memory needed to build it up from a list of words. We will show later how compression of the structure keeps its size in considerably small values, thus allowing the construction of a dictionary from a big lexicon in a common computer.

The construction method of the structure is an adaptation of the one shown in [1], so we will refer to this article for more details. However, we want to note that this method keeps a permanently minimized uncompressed word graph in memory. This graph is compressed during storage of the final structure. The construction can be done in $O(n)$. The result of this construction is a really small structure that grants efficient $O(1)$ time for single word search and supports complex queries. In the current application there are two dictionaries, which contain the set of roots or basic Galician words, and a complete set of words and all their valid variations. The second one contains over 11 million terms.

The data structure we designed can be accessed both to retrieve single words or to find some patterns. We have implemented generic algorithms that allow single-word search and search by substrings. Single-word retrieval can be done by searching for a path in the graph corresponding to any of the rotations of the word. Thus, every search for existence of a word can be processed by crossing as many edges as characters it contains. In our vocabulary, composed of real words of strongly limited length, this shall be considered $O(1)$ complexity. Requests for a set of words containing a substring can be done by searching for a particular rotation of the substring and decoding the resulting subgraph, which contains exactly the requested set. Searching the subgraph is immediate, and retrieval time is proportional to the number of words. These algorithms are combined with the Scrabble.gz game restrictions to provide an efficient method to search for available moves in a match. Other specific algorithms were also developed to process other queries required by the application.

4. Evaluation

The application we have presented in this paper is currently in the production phase, so we cannot show here real usage information. However, it has been thoroughly tested for efficiency and usability. Several thousand games were played just to test and adjust the interface, in order to increase the real-time game perception of the users. Some parameters of the application, like the time limit for each turn, have been also adjusted based on results from this tests. The game can currently be played at http://naron.dc.fi.udc.es/scrabble.gz.

On the one hand, the HTML code of the web page occupies 124 KB 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 objects sent by the web server to each client occupy in total less than 1 KB. As was explained in Sect. 3.2, the game information must be frequently updated by each client in order to achieve a real time perception of the game. Our architecture saves most of the bandwidth compared to a normal web application approach. Furthermore, requests are organized so that only elements that may have changed are returned, so the bandwidth needed by our approach is still lower. In fact, the average size of each update is about 500 bytes. The same happens to specific actions of the players, like queries from the dictionary, where only the required information is returned. Suppose an update rate of 4 seconds, as is configured in the final application, and consider an average of 2 players, 30 minutes and 30 special actions by match. This is approximate to data obtained from our testing. A classic web application would transfer more than 100 MB of information only for this match, while our approach would transfer about 500 KB.

Assuming that the communication is a solved problem, the dictionary becomes the application's bottleneck. We have developed a compact data structure to keep the dictionary in main memory. The dictionary stores 11 million words in 12 MB, one tenth the size of the raw word list. Efficiency of queries against this kind of structures has been proved. The only queries that imply a hard computation are those related to move generation. This procedure is only needed in games against the computer and when users require help from the computer. During the tests of the move generation algorithm (performed with the application installed in an Intel Core 2 Duo 2.14 GHz, 2 GB RAM) our algorithm obtained the set of all valid moves in an average time of 15 ms, and the computation time was quite stable during all the game. If we consider that this kind of requests are actually unusual in our game, the cost of computing the request can be perfectly assumed by a normal web server. The processing time is really slow compared to the network latency, so users will never loose the real-time perception. In fact, speed of move generation calculations forced us to slow down the computer's moves to simulate the evaluation time of a real user.

5. Conclusions and future work

We have presented in this paper the architecture and some implementation details of a web application that simulates a Scrabble game where users can play in real-time. This was achieved without using any plug-in or applet, which means that the game can be played without the common problems associated to the installation of these components. We have described how the AJAX philosophy is used in the implementation for the data exchange and how a state-machine in the server is used to synchronize the interaction.

We have selected two Galician language dictionaries and we have designed the data structures such to represent them, in a way that other applications that need to work with Galician words may use them. In fact, most applications that need to work with words (of any language) could use these structures. Completeness of the current dictionary, with several millions of terms, and extensibility of the algorithms to look across it, make of it a very useful development. The architecture of the application allows real-time interaction with the dictionary, including complex queries. Integration of the dictionary with this kind of architecture makes it a powerful tool for its use, for instance, in education, increasing children's language capabilities.

Our lines of future work include an improvement of the game that will allow suggestions to the users to be restricted to a subset of the most common words. This would make the games more understandable, avoiding results that involve complex or very unusual words. Another future development could be the customization of the algorithm for move generation to include heuristics. The current selection method takes the best-scored word, which is enough in most cases, but advanced players take advantage of strategies that could surpass the program's abilities. A different line of future work is the application of the developed dictionary in some other educational web-based systems, following the same philosophy and purposes exposed here. Finally, we plan on updating the communication protocol of the architecture to support new *Reverse AJAX* functionalities. Reverse AJAX includes a mechanism for pushing server data back to the browser and could improve a lot the efficiency of our architecture.

References

[1] A. Appel and G. Jacobson. The world's fastest Scrabble program. *Communications of the ACM*, 31(5):572–578, 1988.

[2] J. J. Garrett. Ajax: A new approach to web applications, 2005. Retrieved from `http://www.adaptivepath.com/publications/essays/archives/000385.php` in October 2007.

[3] S. A. Gordon. A faster Scrabble move generation algorithm. *Software - Practice and Experience*, 24(2):219–232, 1994.

[4] M. R. Luaces, O. Pedreira, . S. Places, and D. Seco. Trivial.gz: A web-based collaborative game to promote Galician culture. In *Proceedings of the 4th International Conference on Web Information Systems and Technologies (WEBIST'08)*, 2008.

[5] L. D. Paulson. Building rich web applications with Ajax. *IEEE Computer*, 38(10), 2005.

[6] World Wide Web Consortium. Extensible Markup Language (XML). Retrieved from http://www.w3.org/XML/ in October 2007, 2007.