Networked Games based on Web Services

Chong-wei Xu and Hongwei Lei

Computer Science and Information Systems
Kennesaw State University
cxu@kennesaw.edu

Daniel Xu

Computer Science
Georgia Institute of Technology
daniel_n_xu@hotmail.com

***Abstract*—On one hand, web services have demonstrated their important roles in the field of computing. On the other, networked games need server support, which is usually based on socket programming. For example, in a two-player take-turn game using TCP protocol, a server communicates and coordinates the two game GUIs utilized by the two players. This gives rise to one important research question, "Can the server take the advantages of web services in order to replace the sockets while supporting networked games?" This article describes some technical aspects for accomplishing this goal.**

***Keywords*—Networked game, Game server, Web services, Socket programming, Game development**

1. INTRODUCTION

Networked games accommodate multiple players and create a new environment for networking and inter-player communication. As a result, the players are able to make friends and enjoy their group activities while enjoying the games.

In general, networked games fall into either of these two kinds: (i) Take-turn games that allow only one player at a time to play the game, and (ii) The asynchronous multiplayer games. In the first kind, the current states of a networked game are exchanged through the network communication facility in a sequential fashion. The second kind allows players to asynchronously play the game while the game states are randomly generated and transferred.

The traditional approach for developing networked games utilizes socket programming [1], [2]. In a two-player take-turn game using TCP protocol, a Server holding the game logic must not only communicate with the two clients that perform the game GUI functions, but also synchronize these players' moves to ensure and enforce that only one player is active, while the other waits. Although every game GUI is implemented as a Thread such that they run in parallel, the actual process in the background remains strictly sequential. Based on the game logic held at the Server, states data is manipulated by the Server for making decisions (e.g., deciding the winner), and some of the states are further transferred to the other side. This implementation is practicable but it inherently couples the client-server units tightly, allows a limited number of players and thus renders the software architecture unattractive from the view points of maintenance and reuse.

Web Services [10], [11] has successfully exhibited the practical significance of its distributed business model. Since the vendors have agreed on common web service standards, these externally available distributed application components can now be employed to integrate computer applications that are written in different languages and run on different platforms. As SaaS (Software as a Service) web services are developed, hosted, and operated by software vendors; acceptable quality and middleware services, including certain degree of security is assured to the users. Under this model, the users are no longer required to own the software, but rather agree (with or without paying) for using it. Making software services available for clients connecting from anywhere at any time, web services play important roles in the field of computing, including those of grid and cloud computing.

A web service is traditionally defined by the W3C as 'a software system designed to support interoperable machine-to-machine interaction over a network'. Web services have generated immense amount of interest in recent times in terms of its applicability in social interactions including gaming scenarios [5], [6]. Networked games do need to be interactive, but, a more important question is whether it is possible to make networked games interactive incorporating web services? Switching from the socket programming paradigm into the enabling technology of web services promises enormous business value for networked games [3]. As such, single user games, such as card games like Blackjack has been implemented by using web services technology [4]. Here we study both the web services and the gaming technologies, in order to ascertain whether web services could be aptly engaged in developing networked games.

In this work, we simplistically focus on involving two-player take-turn networked games. That the take-turn games have only a set of fixed number of states to be transferred at a time (which is easier to manage than the games that have random states), renders better tractability and insights in to our experiments without further complicating the analytics of this paper.

2. AN ABSTRACT MODEL OF GAMES

Games fall under multiple categories and genres. In order to extract common features of games, we develop and present an abstract model of games here. Although our abstract model is simple, it provides excellent guidance in developing different kinds of games with novel software architecture.

DOI: 10.5176_2010-2283_1.1.28

Our proposed model of games is based on the classic MVC model, viz. the View, the Model, and the Controller [7]. By design, this architecture separates these three units to eliminate interdependencies, a concern in designing and implementing software.

We look at a game as a software application, consisting of the following three units.

A game = gameGUI + gameLogic + player

In our above depiction, the gameGUI is synonymous to the View, the gameLogic to the Model, and the player to the external Controller, in terms of the classic MVC model. A game may additionally have multiple internal controllers, such as the animation mechanism that moves sprites around, further causing collisions among sprites and thereby generating new game states. Also, a certain threshold of some other properties (e.g. the speed of the sprites can be dependent on the current score of one or more players in the game) can also play the role of internal controllers.

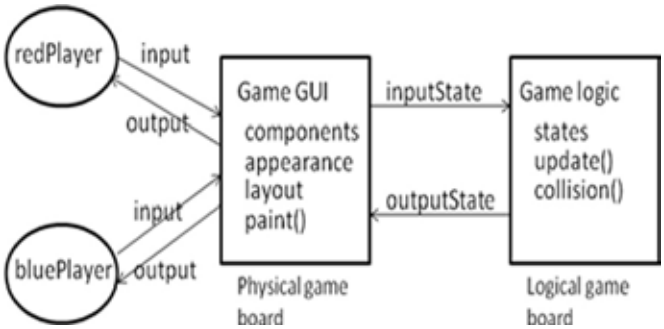


Fig. 1. The abstract model of two-player games.

For stand-alone two-player games, which are played by two local players, the relationships among these units in our abstract model are shown in Figure 1. Thus, the important set of considerations for developing a game is to find out (i) what input states and output states should be passed between the gameGUI and the gameLogic, and (ii) what inputs should be caught from the players and what output should be shown on the gameGUI for the players.

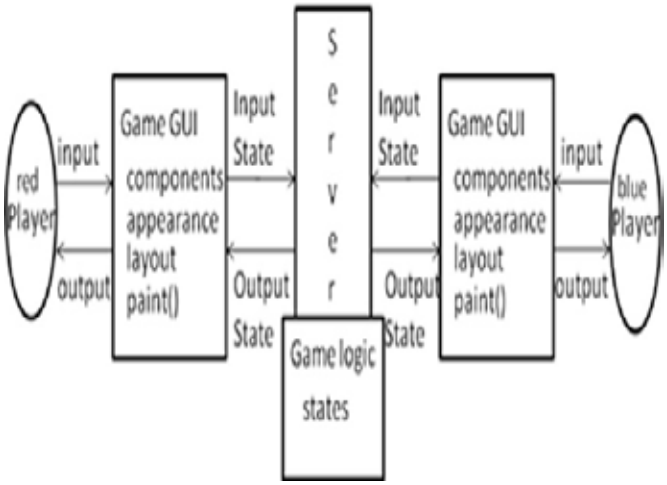


Fig. 2. An abstract model of networked games.

Based on the above model of games, networked games, which are played by two remote players, can be represented as two game GUIs with an additional Server between them as shown in Figure 2. The two GUIs go through a network communication facility to communicate with the Server while the Server acts as the center for both game GUIs, and shoulders the responsibility to apply the game logic for synchronizing two players.

3. A SOFTWARE ARCHITECTURE

The abstract model guides us to investigate novel software architecture. In order to clearly separate the network communication facility from the game GUIs, we have developed a class CommInterface (Communication Interface) that remains sandwiched between the Server on one side and the game GUI on the other side, and works as a communication interface layer as shown in Figure 3.

Clearly, the TCP protocol employs server socket for the Server and socket for the CommInterface so that the Server and the class CommInterface remain physically separated. In order to separate the CommInterface from the game GUI, we have designed another interface, entitled as GamePanelInterface, which contains abstract methods for transferring game states and is implemented by the class GamePanel in the game GUI so that the communications between the CommInterface and the Game GUI are based on interface-to-interface structure. This new architecture remains the class CommInterface "stand-alone" and be completely re-usable for similar networked games. The only modifications that a game may need would involve the GamePanelInterface.

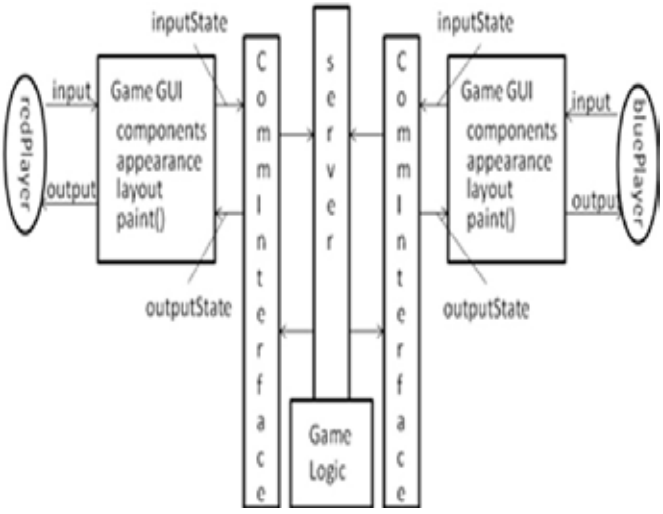


Fig. 3. A software architecture of networked games.

4. A CASE STUDY: A WEB SERVICES BASED NETWORKED GAME TICTACTOE

The game TicTacToe is a simple game played by two players. We select this game as an example for explaining what we have done because (i) it is a well-known game that does not need detailed descriptions, (ii) it is a take-turn game with two players, and (iii) its networked version with socket programming are available without much difficulty

[8], [9]. By using it as our example, we can compare the web services with that of the socket programming based games in order to further explore the differences between these two enabling technologies. In addition, insights garnered from this simple example can be utilized to construct suitable software architecture for more complicated games.

Obviously, the implementation st arts from a stand-alone game TicTacToe, specifically for two different game tokens on a 3x3 game board. Upgrading it to a networked game requires one to identify what input and output states should be exchanged through the communication channels. Clearly, the input states from the game GUI to the Server are the values of the row and the column that the player clicks on the game board since the game logic in the Server needs these data to decide the winner. These data should be further passed to the other game GUI to display the new token that was just added on the game board for both players. In order to maintain the states of the game in the game GUIs and the Server, three 2D array data structures corresponding to the game board need to be constructed in each of the game GUIs as well as the Server. The output states from the Server to the game GUI should include, (i) 'who takes turn' for synchronizing two players, (ii) 'who should have the token' for distinguishing the two players; and (iii) 'who is the winner' for judging and finishing the game besides the values of the row and the column of the new token sent by the other side. Clearly, the above (four) data requirement necessitates four separate methods to be defined in the GamePanelInterface.

The next step is to design and implement the web service, which we call TictactoeWebService, to be the Server for the networked game, which provides a set of operations to accept the input states and generate the output states. The web service client, TictactoeWSClient, is a java SE program that includes the interface 'commInterface'; and the game GUI, which contains the GamePanelInterface. The class CommInterface extends the class Thread that communicates with the web service by accessing operations defined in the web service and also communicates with the game GUI through the GamePanelInterface.

The major actions in running the networked game are depicted in Figure 4: One of the game clients connects to the Server first, to which the Server assigns 'player1' and issues a token. When the other game client connects to the Server, it receives an assignment 'player2' and its token. Thereafter, the Server gives the authority to player1 to start his/her game. After player1 plays the game, the resulting data are the coordinates of the new token just placed on the game board. The coordinates are then sent out from player1 to the Server. When the Server receives the data, it switches the turn to player2. Player2 now places his/her new token on the game board, and similarly as before, the coordinates of the new token are then sent to the Server and eventually to player1. This process is repeated until either one of them wins the game or the game ends up with 'draw'.

A closer look tells us that the software architecture can be further simplified. The two decisions 'who takes a turn' and 'who is the winner' made by the Server can actually be moved to the game GUIs. The reason is that the Server determines 'who takes a turn' when it receives the coordinates of the new token just placed on the game board. In reality, the coordinates of this information are known by both the game GUIs themselves. When one side places the token on the game board, it knows it should stop the turn; when the information is mirrored on the other side, the other side knows it can take the turn. Thus, the Server can be released from the responsibility of decision making in the first place. We can ask the game GUIs to make the decision by themselves. Only when the game starts, the Server needs to make the initial assignment 'who takes a turn' once. By similar argumentation, it is apparent that decision 'who is the winner' can also be judged by either side separately if we move the game logic from the Server to each game GUI respectively.

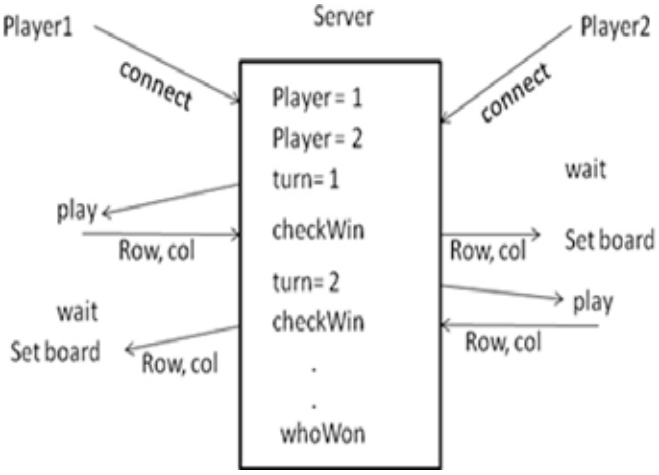


Fig. 4. The actions in the networked game.

In fact, a stand-alone game originally employs both game GUI and game logic. We are not necessary to move the game logic into the Server. That is, we could keep the two sides in the networked game remains the same as the stand-alone version, which not only avoids the modification of the original game code but also reduces the number of states that should be transferred. Adoption of the above strategy reduces the model of networked games with a new architecture as in Figure 5.

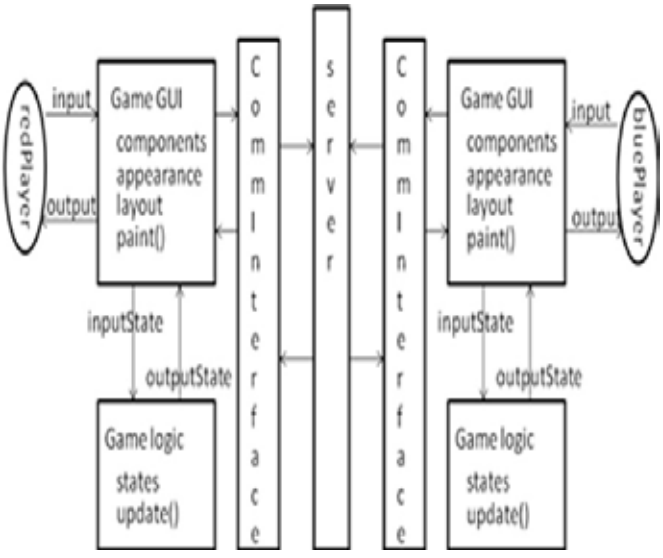


Fig. 5. The final model of a networked game with new software architecture.

Based on this new strategy, the GamePanelInterface has only three abstract methods remaining: setTurn(), setToken(), and setOther(). The method setTurn() and setToken() are used only once to pass the take-turn and the token parameters from the Server to the gameGUI when the game initially commences. The method setOther() is used to transfer the coordinates from the commInterface to the gameGUI regularly during the game. Therefore, both the web service and the class CommInterface become a pure network communication facility used solely for data transfer with no responsibility to make in-game decisions.

The web service plays the role of a Server with a very simple body of code. It only contains the following six operations: connect(), getPlayerStatus(), getPlayerArrival(), getColumn(), getRow(), and setCell(). The class CommInterface invokes these operations to get information from the Server to catch the states of the other side.

Note that this software architecture keeps the game body nearly as original. The only difference lies in the requirement that the GamePanel needs to implement the GamePanelInterface. The web service stands simplified without any game logic. The class CommInterface only communicates with the web service and interacts with the GamePanelInterface. All of them do not bear any responsibility to deal with anything specifically pertaining to the game. Consequently, our proposed software architecture becomes a framework that provides web services as the Server and the classes CommInterface and GamePanelInterface for forming a communication facility so that any this kind of games can be easily adapted as a networked game without modifying the original stand-alone game.

5. TECHNICAL COMPARISON WITH SOCKET BASED SERVER

As we have already mentioned, web services offer significant advantages because they are distributed application components and are externally available. One can use them to integrate computer applications that are written in different languages and run on different platforms.

Switching the enabling technology from socket programming to web services for implementing networked games brings other attendant challenges. The significant differences between them stem from the fact that the Server in socket is an active object while the web service is a passive one.

TCP socket support input and output channels, which provide two methods readInt() and writeInt() for data transferring. The readInt() call can automatically pauses the Server for waiting the transferred data to come from the CommInterface and the writeInt() call can send data out to the clients. In other words, the readInt() and writeInt() automatically perform synchronization functions. On the other hand, being passive objects, web services can accept inputs and do computations, but cannot send data out. A client cannot 'sit and wait' for data, it has to probe the web services Server continuously for getting required data from the Server. Thus, the client needs to use the Thread sleep() method to pause itself and keep checking the web services Server with a while loop to ensure the data arrival, as demonstrated in the following code.

```
private synchronized void waitForReceiving (int player)
throws InterruptedException {

if (player == PLAYER1) {
    while (port.getPlayerArrival() != PLAYER2) {
        Thread.sleep(100);
    }
    try {
        receiveInfoFromServer();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
} else if (player == PLAYER2) {
    while (port.getPlayerArrival() != PLAYER1) {
        Thread.sleep(100);
    }
    try {
        receiveInfoFromServer();
    } catch (Exception ex) {
          ex.printStackTrack();
    }
}
```

Additionally, web services have been a mature and widely used business model, and one can access web services from anywhere at any time unlike the socket based approach, which can only live in a certain server. Also, web services are anyway designed to support multiple players, thus it becomes easy for both developers and players to work with without worrying whether the Server is a sequential or concurrent as in socket programming.

6. EXTENSION OF THE FRAMEWORK TO OTHER GAMES

The benefits of the above framework may be easily extended to other games. For example, the classic game Connect4, which is somewhat similar to chess, has a 6x7 board, and accommodates 2 players, one plays red pieces and the other plays blue pieces, who take turn in the moves. Each of the players moves an arrow pointing to a column before dropping his/her piece onto the column. Once a player is able to place 4 pieces consecutively in one consistent direction (the x-direction, the y-direction, or diagonally) wins the game. The stand-alone version of the game Connect4 is implemented in both Java and JavaFX. The UML diagram of the game Connect4 in Java is shown in Figure 6. Even it is implemented in different languages, we have successfully applied the above software architecture to implement the game as a web services based networked game, vide Figure 7.

7. CONCLUSION AND FUTURE WORK

Computer gaming is of general interest today, commands wide scale appreciation and boasts of rich content and quality

presentation. These games merge our understandings and capabilities in Humanities, Arts, Mathematics, Physics, Graphics, Animation, AI, multimedia, and programming technologies, they also constantly challenge our problem solving capabilities. Due to the pervasive utilization of web services, developing networked games based on web services not only promises another platform for gaming in general, but also brings interesting games and their intelligent adoption into the coursework for teaching web services. We hope that our current research provides another impetus to this interesting development and merger of web services and networked games in myriad search and understanding of knowledge.

Here we have presented our initial research of two rudimentary implementations of web services on TicTacToe and Connect4. Going ahead, we may add more interesting features with the games, and test their successful adoption of web services. For example, we may extend these games to allow multiple players to join the game, (i) allowing only specific pairs to play against each other or (ii) allowing entrants to join the game as observers or judges, or (iii) as specialist observers, who could support and provide real-time suggestions for a current player (the final decision could still belongs to the real player). Reusing the Server, as well as the classes CommInterface and GamePanelInterface in the framework, we are reasonable certain that we could develop more networked games such as Othello. At a more challenging level, we also plan to extend this research to design and implement action games, especially games with random actions, such as Snooker.

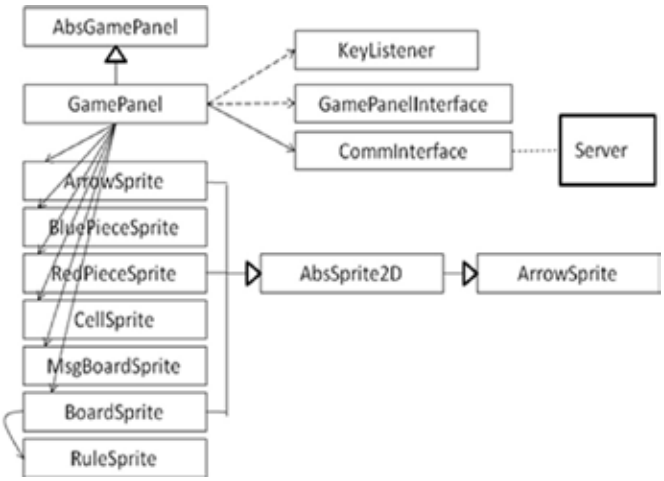


Fig. 6. The UML diagram of Connect4 in Java.

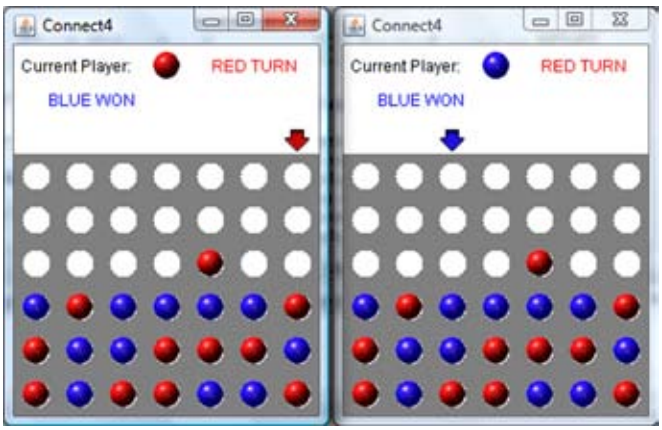


Fig. 7. The networked game Connect4 implemented through web services.

REFERENCES

[1] Fan, J., Ries, E., and Tenitchi, C., (1996). Black Art of Java Game Programming, Waite Group Press.

[2] Morrison, Michael, (2005). Beginning Mobile phone Game Programming, Sams.

[3] Sharp, Chris, (2003), Business Integration for Games: An Introduction to Online Games and E-business Infrastructure, DOI = http://www.ibm.com/developerworks/webservices/library/ws-intgame/.

[4] Deitel, P.J. and Deitel, H.M., (2008), Internet & World Wide Web How to program, 4/e, Prentice-Hall.

[5] Rubel, Steve (2008), The Future is Web Services, Not Web Sites, DOI = http://www.micropersuasion.com/2008/03/the-future-is-w.html

[6] Ellis, Steve (2008), The Future is Interactive, Not Online, DOI = http://thenewmarketing.com/blogs/steve_ellis/archive/2008/03/17/5467.aspx.

[7] Model-view-controller, DOI = http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

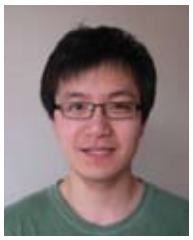
[8] Liang, D. (2008), Introduction to Java Programming, 7/e, Prentice-Hall.

[9] Jia, X. (2002), Object-Oriented Software Development using Java, 2/e, Addison-Wesley.

[10] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004) Web Services, Springer.

[11] Deitel, Deitel, Gadzik, Lomeli, Santry, Zhang (2003) Java Web Services, PH-PTR.

Dr. Chong-wei Xu is currently a Professor of Computer Science in Department of Computer Science and Information Systems at Kennesaw State University. He received his Master in Computer Science from University of Wisconsin-Madison and his Ph.D. in Computer Science from Michigan State University. His current research interests mainly include Internet and distributed system technologies and gaming technologies.



Daniel Xu currently is a double-major senior at Georgia Institute of Technology. After he received his bachelor degree in Electrical Engineering, he joined the Computer Science B.S. program. His research interests include database and gaming technologies.

Hongwei Lei received his Masters in Applied Computer Science from Department of Computer Science and Information Systems at Kennesaw State University. He has 3-years industrial working experience in software development and a former graduate research assistant working on several research projects. His current research interests include gaming, 3-D graphics, and web technology.