Real Time Graphical Chinese Chess Game Agents Based on the Client and Server Architecture

Peter Vo, Yan-Qing Zhang, G.S. Owen and R. Sunderraman

Department of Computer Science
Georgia State University
P.O. Box 4110
Atlanta, GA 30302-4110 USA

Abstract. The client and server architecture is currently widely used in industry; therefore, it is worthwhile to perform further investigations into its usefulness in different applications. To accomplish this, this paper will demonstrate its appropriateness by implementing an Internet based Chinese chess game using client and server architecture. This paper also has value for developers who would like to develop similar applications, such as Western chess, Internet Relay Chat, etc. In implementing the game application, the server program is developed with Java technology while the client program is implemented in C++ with the help of MFC to facilitate the development of a 2D graphical user interface. In the future, the client program can be modified to be a web application. Importantly, the Internet-based graphical Chinese chess agent system can be used to teach students to understand intelligent agents and game playing in an artificial intelligence class, networks, graphics and other relevant techniques in other computer science classes.

1 Introduction

Computer technology has been extensively developed and a large quantity of software has been written to solve complex business and entertainment issues. More and more features, such as increased business functionality and graphical user interfaces (GUIs), have been added to these software systems. To accommodate user demands and maintain system efficiencies in a timely design and implementation enhancement cycle, software developers frequently have questions related to maintainability, usability, flexibility, interoperability, and scalability.

In a monolithic application model, with business logic, database access, and GUI all combined in one module none of the 5 essential considerations is satisfied. While client server architecture has been developed relatively recently in the history of computer technology it has been proven that client server architecture is an essential part of any enterprise level application, and it satisfies the 5 considerations.

A GUI is a basic component for any visual application ranging from pre-school educational software to sophisticated professional enterprise applications. For pre-school students, it can be used to display simple lessons and educational and enter-

taining games. Currently GUIs are 2D with some exploratory research into 3D GUIs. For business applications 2D graphics is a basic component for the complex GUI and is used, rather than 3D graphics, for both computational and design simplicity and for better system performance.

Since "client/server" is a well-known successful architecture, and 2D graphics is a basic and popular method for building a GUI application, these techniques are employed in this work on a Chinese chess application. This agent system consists of 3 components. The first is the server program, where game logics are validated and players' information is manipulated. The second is the client, which is the GUI for interacting with the server. The third is a Chinese chess auto-player program, which acts as an intelligent agent.

The rest of the paper is organized as follows. Section 2 discusses the Chinese chess software system in terms of its functional design specification as well as the detailed design. Section 3 discusses the Chinese chess auto-player agent with its functional design specification and the detailed design. Section 4 shows major technical merits and applications in education, artificial intelligence, networks, graphics, etc. Finally, Section 5 gives conclusions and future work.

2. The 2D Client-Server-based Chinese Chess Software

The client/server architecture shown in Fig. 1 is a logical extension of modular programming technology in which a monolithic piece of software is separated into 2 constituent components. One of the two is called the "Server" and the other is called the "Client". The server in this architecture is defined as a service provider, and the client is defined as a service requester. In essence, the client/server architecture does not require that the client and the server have to be on the same memory address space, or on the same machine. This feature is a solution to many enterprise applications. On the other hand, the separation greatly benefits both development and maintenance of the software.

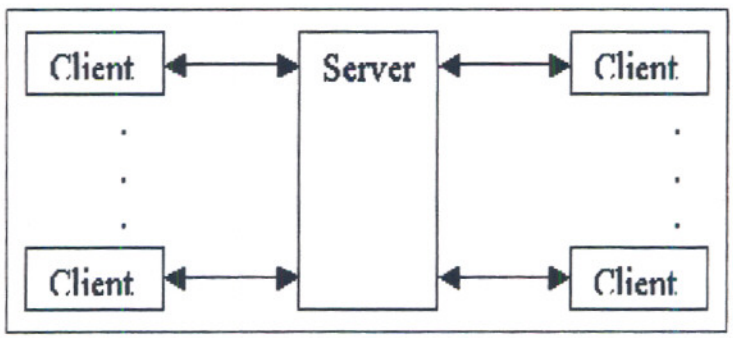

Figure 1 The Chinese chess game architecture

A client program is a service requester that sends each request to a server program and, if necessary, waits for the data response from the server. The most common role of a client is to manage the user interface components of the application. Because of this popular role of the client program, it is also referred to as the presentation layer of the client/server architecture. Before sending a request to a server, a client usually validates data entered by its users and performs some proper business logic.

A server is a program playing a role as a service provider. It is always running and waiting for a client request. Whenever it receives a request from a client, it performs the specified task and returns the result, if any, to the client. The most common role of a server is to manipulate the database for the application. It is also very common that a server program provides logic and computations for the application.

Every Internet application can be developed using client/server technology. The Internet Chinese Chess system is an example of this and the client/server architecture is a perfect model for this application. The integral part of the Internet Chinese chess application is the server, which resides on a fast computer capable of handling many simultaneous connections. Before any client connects to it, the server is running and waiting for client connections to be established. When there is at least one client connecting to it, the server also listens for any request from other potential clients. For any specific request, the server performs the task and returns the result to the corresponding client. For this application, the client is responsible for displaying the visual representation of the chess board to the players and providing a GUI to facilitate its user's interaction with the server. Figure 1 demonstrates the relationship between the server and clients.

The client program is developed using Visual C++. The server program is developed using Java [4, 5, 8] and SQL [13], therefore it can be run on any operating system with JVM installed.

Each instance of the client program runs on a player's computer. The client program's goal is to facilitate and fascinate players in communicating and playing games with others online. The client GUI comprises 1 menu bar, 1 toolbar, and 1 splitter window. Figure 2 shows the client program GUI. The menu bar consists of File, Edit, View, and Help submenus. The File submenu has 3 menu items: Print, Print Setup, and Exit. The Edit submenu has 6 menu items: Undo, Redo, Copy, Cut, Paste, and Delete. The View submenu has 2 menu items: Toolbar and Status bar. Finally, the Help menu contains only 1 menu item: About. The toolbar consists of 16 buttons. Each button corresponds to a command option supported by either the server or a client. Finally, the splitter window consists of 2 panes - an upper pane and a lower pane. The upper pane is for displaying the output response from the server while the lower pane is for the user to issue their text commands to interact with the server.

2.1 Server Detailed Design

The first time it runs, the server needs to create 2 tables: players-info and games-info. The players-info columns are: name, password, score, balance, email. The games-info columns are: game-name, game-num,, e.g., game0, game1, ... , and game9.

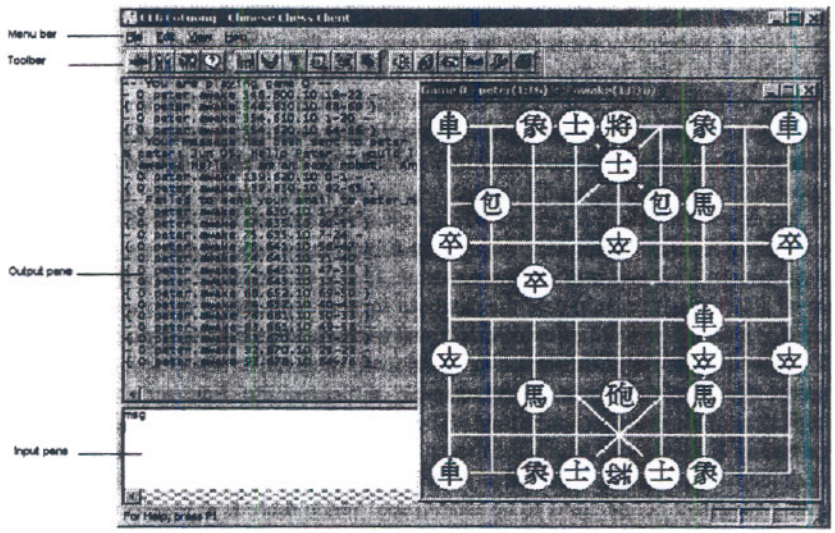

Figure 2 The client GUI

Anytime the server starts, it creates one and only one database object and connect to the database for synchronization. It also creates new types necessary for the server to operate: "Player-Record", "Game-Negotiation", and "Game-Record".

To speed up performance, lists are implemented using a Hash Table [1].

The server waits for a connection by infinitely looping and checking for new connections. When there is a connection request from a client, the server accepts the connection and spawns a new thread to communicate with that client. After the new thread has started, the server goes back to its waiting state and is ready to accept a new connection [7]. The following state diagram in Fig. 3 sketches the server's connection handling.

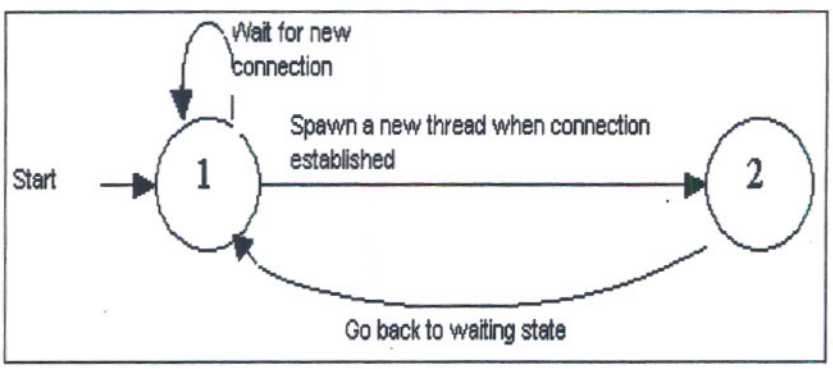

Figure 3 A sketch of server's connection handling

Algorithm:

1. Create a new socket and wait for a new connection.
2. Establish the connection when there is a connection request.
3. Create a new thread (player-thread) to communicate with the client. The main thread then goes back to 1.
4. The player-thread will then check for the player's existence in the database. If the player is already in the database, it will validate the player's user id and password. Otherwise, it will guide the new player to register as a new member and collect all necessary information required to fill out the Player-Record. Then it saves all information in the Player-Record except for thread-id into the players-info table. The Player-Record will then be added to g-players-records in memory to be ready for fast collection of player information. Some fields in Player-Record do not have any value for a first time player or a new login player. The server uses some default values as described below:

   Score: 1500,
   obs-number: -1,
   game-number: -1,
   idle-seconds: 0,
   balance: 0.

2.2 Client Detail Design

The client is the GUI of the application. From the players' point of view, the client facilitates their interactions with the server by providing an output window, an input window, and a chessboard representation. The input/output windows and the chessboard are the elements we are primarily concerned with and we used the Visual C++ [2, 12] wizard to generate the main window (CMainFrame), the menu bar (CMenu), and the toolbar (CToolBar) [10, 15].

2.2.1 CMainFrame

CMainFrame encapsulates the main window of the client. It contains the menu toolbar, and the splitter window, which has two child views – the input and output view. Any event generated when user clicks on a menu item or a toolbar button is handled inside this class.

2.2.2 Create the Splitter Window

We declare member variable m_splitter, m_input, m_output in CMainFrame. Then override CMainFrame::OnCreateClient() and create a splitter window and attach it to m_splitter. After the creation of m_splitter, invoke its CreateView() and GetPane() to assign the m_input and m_output to the result view. From now on, the m_input and m_output are the handles to the input window and output window. Whenever the client receives input from the server, it updates the text in the m_output window. The client interprets the text in the current line of m_input window as a command to interact with the server.

2.2.3 Connect to the Server

Without a connection, the user cannot interact with the server at all. The connection to the server can be done with the window socket protocol. With the help of MFC, a socket connection is no longer a difficult problem. The steps necessary to establish a connection are described below:

1. Create a member variable in CMainFrame name m_socket.
2. Create the socket whenever user requests to connect to server. Then invoke AsyncSelect(FD_READ | FD_CLOSE) to register for input and close events
3. Override the socket's OnReceive() to be informed when there is some input to be read.
4. Override the socket's OnClosed() to be informed when the connection is closed.

3. The 2D Internet-based Chinese Chess Auto-Player Agent

There are times no one is connected to the chess server, for example, when the server is first started. The first player who connects to the server will be there alone. In this case, the player cannot play any game or chat with other online players except for querying players' information. That would not be a lot of fun at all. Therefore, the server should have an intelligent agent that can imitate a player to play chess with others.

3.1 Intelligent Software Agent

An intelligent software agent, also known as a software agent, is a piece of software that acts as robot. An intelligent agent is an agent that can make some autonomous decisions. It can handle some tasks involving in making decision and act as smart as human in some situations [6, 14]. For example, an agent can be designed to watch for a specific stock symbol's price. I can make some decision that if the stock price goes up to some value, it automatically sells the stocks. On the other hand, it may notify the owner when the stock price goes down to a very low value. In the game industry an intelligent agent in a game is usually called a Non-Player Character or NPC.

3.2 Auto-player's Functional Design Specification

The auto-player is a separate process that runs on the server machine. It is equivalent to a client and a player on that client. Therefore, after the auto-player was logged onto the server, from another player's point of view it is a player. The auto-player also greets each player anytime he/she logs onto the server. Then, it automatically offers the player a game. If the player accepts the game, it will play with him/her as a real player. For this simple intelligent agent, it thinks only 3 levels in depth while playing a game.

The auto-play
dow. The menu
has 2 menu items
The toolbar has 4
splitter window h
text sent from th
administrator) to i

3.3 Auto-player'

The GUI detail d
that of the client
the robot will not
robot is the board

3.4 Min-Max Te

Suppose that our
the best move sec
the robot has N1
have N2 possible
turn may have N
At level 1 the ro
when its opponen
again, it wants to
Max technique.
O(nlog n) sorting
artificial intellige
[9].

3.5 Robot's Boar

Although there ar
niques are built
simple robot's be
Max search.

The best mo
quently, it needs t

a. Data typ
   Fro
   To(i
   Sco
b. Global v
   g-B

The auto-player GUI is comprised of a menu bar, a toolbar, and a splitter window. The menu bar comprises of 2 submenus: "File" and "Help". The File menu has 2 menu items - connect, exit - while the Help menu has only menu item - about. The toolbar has 4 buttons in order: Connect, Who, Refresh, and Activate. Finally, the splitter window has 2 panes an upper and a lower pane. The upper pane is for output text sent from the server. The lower pane is for the user (usually the server's administrator) to interact with the server by issuing command lines.

3.3 Auto-player's Design

The GUI detail design of our Chinese chess auto-player is very simple and similar to that of the client program. However, the auto-player has less command buttons since the robot will not need those buttons. The detailed design we need to focus on for the robot is the board model used to perform the best move calculation.

3.4 Min-Max Technique

Suppose that our robot thinks 3 levels in depth. This means that the robot will pick the best move sequence in all possible move sequence. Let's suppose that at level 1, the robot has N1 possible moves. For each move in N1 of the robot, its opponent may have N2 possible moves. Finally, each move in N2 of the opponents, the robot in turn may have N3 possible moves (thus there are roughly N1*N2*N3 possibilities). At level 1 the robot wants to select the best move for itself. However, at level 2, when its opponent moves, it wants the opponent to have the worst move. For its turn again, it wants to select the best move. This process of computation is called Min-Max technique. Beside the Min-Max technique, we can use Alpha-Beta prune with O(nlog n) sorting algorithm [3] to cut off the calculation. Moreover, we can utilize artificial intelligent to help the robot make decisions closer to a human's decisions [9].

3.5 Robot's Board Model

Although there are many tricks to make a robot compute a best move faster, all techniques are built up from the Min-Max search in a game tree. For this reason, our simple robot's best-move calculation is designed and implemented based on the Min-Max search.

The best move calculation is a fairly complex and lengthy computation. Consequently, it needs the following data structures and variables.

a. Data type "Move" consists of the following members:
   From(integer): move's "from position",
   To(integer): move's "to position",
   Score(integer): move's score.
b. Global variables:
   g-Best-Moves[]: a list of "Move" objects

g-Robot-Score: robot's score at any time,
g-Regions[]: an array of 90 squares in the Chinese chess board. The values of the elements identify the pieces in the board. For example, if g-Regions[1] = 'red rook', then a red rook is occupying square 1 in the chess board.
g-robot1[]: robot's score at level 1,
g-robot2[]: robot's score at level 2,
g-robot3[]: robot's score at level 3.

c. Best move calculation algorithm
   a. Get all robot's possible moves
   b. For each move 'i' in 1., make-a-move
   c. Get all opponent's possible moves
   d. For each move 'j' in 3., make-a-move
   e. Get all robot's possible moves
   f. For each move 'k' in 5., make-a-move
   g. Get robot's score and store in g-robot3[k]
   h. Unmake-a-move (this will put back the move in 6)
   i. End "for loop with variable k"
   j. Get max of g-robot3[k] and assign the max value to g-robot2[j]
   k. Unmake-a-move (this will put back the move in 4)
   l. End "for loop with variable j"
   m. Get min of g-robot2[j] and assign the max value to g-robot1[i]
   n. Create an instance of "Move" with information consists of g-robot1[i] and the corresponding move information. Then add this instance to g-Best-Moves[].
   o. Unmake-a-move (this will put back the move in 2)
   p. End "for loop with variable i"
   q. Navigate through g-Best-Moves[] to search for the "Move" object that have the best score. The move associated to that score is the best move that the robot should move.

d. Make-a-move: is a function responsible for updating g-Regions[] and g-Robot-Score. It calculates g-Robot-Score by "adding to/subtracting from" g-Robot-Score the piece's value at the "moved-to" position if it is a "robot's move/opponent's move" respectively.

e. Unmake-a-move: this function can be called only if make-a-move function has been called. The purpose of this function is to put back the changes made by a call to a previous make-a-move. Thus, "make-a-move(), make-a-move(), unmake-a-move(), un make-a-move()" will not alter the board model at all.

4. Technical Merits and Applications

Currently, there are many websites supporting games including Chinese chess. This paper discusses a MS Windows based implementation of the game with an auto-

player. Therefore, it has some advantages and some disadvantages compared to the existing ones.

Some advantages of this work compared to the previous ones are:

a. The client is a MS Windows based program. Therefore, its response time is very much faster (possibly up to 3 times) than the web clients.
b. The auto-player amuses even the first login player.
c. Player can move the piece by clicking and dragging the piece naturally.

Main applications include (1) education in artificial intelligence, for example: intelligent agents, game playing, etc., these topics are not easy for students to understand, and (2) other educational applications in computer science such as networks (client-server architecture), and (3) other classes.

In general, distant students can use the online agents similar to the game agent to play with them, learn new knowledge, and create novel techniques conveniently and efficiently. Instructors can also use the smart Web agents to provide students with vivid demonstrations and online experiments like inline games. In the future, real-time online 3D graphics-based teaching and learning systems will play an important role in terms of quality, efficiency, visualization and intelligence.

5. Conclusions

Client/server is a perfect model for internet/networking enterprise applications. It is especially good for applications that support many simultaneous users. In this model, the server program is installed on a fast separate machine with an operating system that can support a large number of processes and threads. The client program has many instances running on different machines and asking for services from the server. This model has also been proven as a cost-effective solution to enterprise level applications. Although client/server is just one of many approaches to distributed computing, it has been frequently used and has dominated many other enterprise solutions.

A 2D GUI plays an important role in any application. For pre-school students, it can be used to display simple lessons and entertaining and educational 2D games. For other uses, especially in business, 2D graphics forms the basis of a GUI for all applications to increase performance and usability.

Currently, the application connects to the database (MSDE) using ODBC-JDBC driver, which is fairly slow. If we can find a JDBC driver for it, we can re-implement the database object for better performance. The application handles connections from clients using the multi-threading feature of JDK 1.3.1. Although threading is cheap, it is still more expensive compared to non-threading. JDK 1.4 supports Socket Channel that we can utilize for our multi-connection application without creating many threads. This will greatly improve our system scalability. Importantly, the Internet-based graphical Chinese Chess agent system can be used to teach students to understand intelligent agents and game playing in an artificial intelligence class, computational intelligence [9] class and networks and graphics in other computer science classes conveniently and efficiently.

Acknowledgements

The authors would like to appreciate the support by NSF under Grant IIS-9980130 and the ACM SIGGRAPH Education Committee.

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman.: Data Structure and Algorithms, Addison-Wesley, Reading, MA (1987).
2. Marshall P. Cline, Greg A. Lomow.: C++ FAQs, Addison Wesley, Reading, MA (1995).
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest.: Introduction to Algorithms, MIT Press, Cambridge, MA (1990).
4. David Flanagan.: Java in a Nutshell, O'Reilly, Sebastopol, CA (1997).
5. David Flanagan.: Java Examples in a Nutshell, O'Reilly, Sebastopol, CA (1997).
6. Tim Finin and Yannis Labrou.: UMBC Agent Web.< http://agents.umbc.edu/> (2002).
7. Elliotte Rusty Harold.: Java Network Programming, Oreilly, Sebastopol, CA (1997).
8. Philip Heller and Simon Roberts.: Java 2 Developer's Handbook, SYBEX, Alameda, CA (1999).
9. J-S. R. Jang and C.-T. Sun.: Neuro-Fuzzy and Soft Computing, Prentice-Hall, Inc., Upper Saddle River, NJ (1997).
10. David J. Kruglinski.: Inside Visual C++ 6, Microsoft Press Redmond, WA (1997).
11. Joseph O'Neil.: JavaBeans Programming from the Ground Up, Osborne/McGraw-Hill, Berkeley, CA (1997).
12. Herbert Schildt.: C++ from the Ground Up, Osborne McGraw-Hill, Berkeley, CA (1994).
13. Ryan K. Stephens, Ronald R. Plew, Bryan Morgan, and Jeff Perkins.: Teach Yourself SQL in 21 Days, SAMS, Indianapolis, IN (1997).
14. Katia Sycara.: Carnegie Mellon University's Robotics Institute. <http://www-2.cs.cmu.edu/~softagents/intro.htm> (2002).
15. Viktor Toth.: Visual C++ 4 Unleashed, Sams Publishing, Indianapolis, IN (1996).

DIMS:
prototyp

[1] LA
Cambresis

[2] Schc

[3] Centre U

Abstract: Tl
a multitude
available on
Distributed
information i
efficiencies t
JDBC-ODBC
data and use
integrate the
metadata to e
flexible way
design our sc
implementin

Keywords: i

1. Introdu

With the r
attention ha
heterogeneou
technology o
and semantic
need to acce
applications
great deman
complementa
Internet, pro
very importa