8-1999

# Distributed Chess Game Using Java and RMI

Christopher M. Jaynes
*Southern Illinois University Carbondale*

# Distributed Chess Game Using Java and RMI

When trying to determine a topic for my thesis project, I tried to keep two main factors in mind. I wanted to choose a topic that would help me learn as much as possible about the topic, and that would also be at least somewhat useful. I had recently been working on a project for another class that was written in the programming language Java. I enjoyed programming in Java, and also enjoyed the object oriented development scheme that is used by the language, where physical or logical objects are represented in your program by independent program modules. I talked with Dr. Robert McGlinn in the CS department about my desire to learn more about Java, and he introduced me to RMI, Remote Method Invocation. RMI is just one implementation of a concept known as object reference brokering. This is an object oriented programming concept in which two programs running on two different computers can make references to each other's components. This makes each program capable of interacting with certain parts of the other program, without having to duplicate the entire program on both computers. Dr. McGlinn and I started discussing projects that I could work on that would use Java and RMI. I decided to write a chess game because I have been a long time fan of the game, and because the complexity of the game would allow me an excellent opportunity to get more familiar with Java. A chess game also lent itself quite well to the RMI concept, as each player could run the chess program from a different computer, and simply make references to each other through a server.

I had learned the basics of object oriented development in earlier courses, so the basic design concepts were familiar to me. I started this project by getting out my chessboard and trying to determine what objects I was going to need to represent in my program. I made a list of all of the objects involved in a chess game, and divided them up and categorized them. I envisioned a board object on which rested various types of piece

objects, such as kings, bishops, or pawns. A board could be represented as a matrix with eight rows and eight columns. If each piece were given an index number, those index numbers could be stored in the matrix that represented the board. Then I had to develop relationships between the different objects. These could be represented by methods contained in the objects. For example, the board would have to know how to move pieces around, so it would have a movePiece method. You tell it which row and column you want to move a piece from, and which row and column to move it to. Each piece would be aware of its own rules of movement, so the board would have to check with that piece to make sure that the move was valid. Each method would take a set of inputs, and then return some kind of output, or perform an action. All of the relationships like this had to be worked out ahead of time, so that the objects could later be seamlessly merged into a working project. Unfortunately, finding all of those relationships and determining all of those methods is much easier to do after the program is completed.

Once I felt I had a fairly comprehensive working model to build upon, I began my implementation. I gathered up several Java books from the library and the bookstores, bookmarked several Java tutorials on the Internet, and read through all of the RMI documentation Dr. McGlinn could give me. The first step was to implement the client-server relationship needed for RMI to function. I created two dummy programs, a client and a server, which could simply connect to one another over a network, and send messages back and forth. Then I added the chessboard matrix and a few piece objects to the server program. The client programs could send messages to the server to move pieces on the board, and the server could send the piece locations back to the client. All of this was just done in plain text, like an Internet chat program. At this point only a few of the pieces were represented, and there were no rules to govern piece movement, but everything was still following my original design quite nicely. Once I had each different piece introduced to the board, I began working on the graphical user interface. I drew each chess piece on the computer using a paint program, and then realized that I had no

idea how to include those pictures in my project. This is where the Internet resources became my most valuable assets. I could find very little about graphics in any of the books I had available. I scoured the net for any examples or tutorials, and was able to piece together little bits and pieces of information about which Java commands and objects are used to manipulate image files. I converted the text messages I had been sending back and forth into usable Java commands that would move little images of pieces around on the screen, and voila, several weeks later I had a working user interface. I could move a piece around on the client program using the user interface, and the server would register the move and update the actual game board. Unfortunately, it quickly grew boring to move pieces about on the board randomly, especially with only one player, so I had to move on to the next step, and begin implementing the rules of the game.

It was in implementing the rules of chess that I realized that my planning had been done a little too hastily. I had overlooked how complex the game really is, and how many nuances there are to capture in order to really re-create the game in the digital realm. I found myself writing several dozen lines of code to implement individual rules. A king moves a certain way, and a rook moves a certain way. Rules like that are fairly easy to program. You can figure out the mathematical relationships between the rows and columns you are trying to move between, and check them against sets of equations. I had planned for this. But when a player decides to perform the move known as 'castling' both a king and a rook are involved. This was not part of my original design. I had forgotten that sometimes it is necessary to move two pieces on the same turn. I had also forgotten that castling can only be done if neither the king nor the rook involved had been moved. To implement this I had to go back and add a property to both of those piece objects to keep track of whether they had moved or not. So I had written interfaces that would allow any piece to move according to its own rules, and then had to go back and modify the entire system, practically rewriting it, just to accommodate one special

circumstance. This happened several times throughout the project. I would have a system that worked perfectly with a partial set of rules, and then when I added the next rule to the game, it would change the way the program flowed, and modifications to existing code were almost always necessary. My planning had not been inaccurate, but it had been drastically incomplete.

In the long run my lack of foresight did cost me something. I was unable to implement all of the features that I had originally planned on. I have included almost all of the basic rules of the game, but one rule remains to be implemented. A pawn that makes it to the opposing player's back row does not transform into another piece the way it should. This is an oversight on my part that I did not have time to correct before the project had to be finished. I spent several hours testing the program, and was able to fix several bugs, but Dr. McGlinn gave the program to some students to test, and they were able to find other programming errors, although I have been unable to find and fix them as yet. I also would have liked to make the software generally available, and to have a server running for SIUC students to log on to and play chess, but this too was out of reach due to my own time constraints.

Although I was not able to reach all of my goals, I still feel that the project was a personal success. I definitely enhanced my knowledge of the Java programming language, and even just in general programming concepts. This was really the largest individual piece of software that I've ever written, and it taught me a lot about software development. I also learned about RMI and object reference brokering, which I found very interesting, and should prove to be quite useful.

Most importantly, I learned the importance of proper planning. I had always known it was important to outline a program before I started writing it, but through the course of this project I discovered that the basics aren't necessarily what need to be planned for most precisely. It is the exception to the general rule that makes computer programming difficult. Computers live by very strict rules, and it is when you want to

bend or break these rules that you have to make a special case, and that means extra work for your program to do. The next time I sit down to plan a program, I am going to write out the basic outline of the project as I always do, but then I will try to think of anything that might be at all different from the way I planned it originally.