# Transmitting Avatar Emotions over the Web

A Master Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment of the Requirement for the Degree

Master of Science

By
December 2003
Jing Yuan
Michelle_Yuan@yahoo.com

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

**Dr. Christopher Pollett**

**Dr. Ho Kuen Ng**

**Dr. Sin-Min Lee**

**APPROVED FOR THE UNIVERSITY**

# Transmitting Avatar Emotions over the Web

## ABSTRACT

This report describes our experiences with a facial emotional character agent being used in a network poker game. We first articulate some reasons why facial emotional agents can advance computer human interfaces, frequently used in, for example, AI projects. We list and summarize several human emotional research papers in the area of AI. We then describe the capabilities of our newly designed and implemented human facial emotion agent. This agent is capable of transmitting facial expressions like eye blink, eyebrow up, and eyebrow down etc. Next we describe the software environment used to design and run interactive media systems. We then describe in detail the design and the implementation of applications by illustrating some of the project written code. We also ask friends to play our network poker game that we implemented in the project and get some hands-on experiments with the poker game. Finally, we conclude with a brief look at the facial emotional character agents we designed as part of a network poker game as well as possible future improvements.

# Table of Contents

# List of Tables and Figures

# 1. INTRODUCTION

## 1.1 BACKGROUND

Over the years psychologists have developed a number of encoding schemes for recording human facial expressions and hand gestures [E41], [EF69], [EF78], [M92]. Recently, there have been a number of AI projects, which attempt to use these recording schemes to write programs that can recognize human emotions. The early stages of development of facial animation technology began in 1992 in the Cambridge Research Laboratory and resulted in DECface [W1]. Several books and conference papers on Facial Animation research have appeared [WL94A], [WL94B], [WL95], [FK96], [WLM98], [W92]. Imaginative applications of animated graphical faces are found in sophisticated human-computer interfaces, interactive games, multimedia titles, and most extensively in a broad variety of production animations, computer generated or not. Also work has begun on designing an XML-based language, HumanML, for encoding such emotional and gesture information. Zong, Dohi, and Ishizuka [ZHMI00] have proposed an XML-based language MPML (multimodal presentation markup language), which is SMIL compliant, for giving character agents emotions while they present things like PowerPoint slides. W3C organization also provides X3D that is an XML-based language for displaying 3D graphics.

The goal of this project is to develop a network poker game with 3D-characters able to express emotional information based on their cards. The technology, which is available to do this translation, is style-sheet and can be applied to a character specified in my language to generate a 3D-object in a language like VRML. Messages can be sent and received from this object. Each client will load the Java Servlet file that allows users to enter or choose a game to connect. It then lets the user to choose a player from a list of 3D players for the game and upload associated players' XML files. The server and other clients will need to know which players are selected. The server receives messages sent from each client and sends the messages to the other clients that have the ability to display the emotion specified by the messages. If the given player on the given client does not have the ability to display that emotion, then the server will not send the messages to this client. All clients have the same set up, which contains two frames.

This report is organized as follows: First, the Introduction provides the background of the project and explains the related work and terminology we used in this project, such as XML, VRML, JavaScript, DOM, and Servlet etc. Second, the Design contains a detailed date flow diagram of the design and architecture of the project. Third, the Requirements state the goal of this master project. Fourth, the Implementation describes the detailed code explanations of this project. Fifth, the Deployment argues for the correctness of the poker hand formula that we used in this project and describes user experiences. Sixth, the Acknowledgements thank all the people who supported and helped in this project. Seventh, the Conclusion simply covers each part of the report and ends my master project report with a discussion of possible future work. Finally, we include all references in the References part and project source codes in Appendix A.

## 1.2 RELATED WORK AND TERMINOLOGY USED:

This project allows users to enter a new game or choose an existing game from a drop down list. It lets users choose different images to build up the player's face. After connecting to a game, users can play a poker game such as dealing, calling, folding, raising etc, and, most importantly for this project, send facial emotional messages to display emotion on a 3D VRML face. Screen shots of the interface are in Figure 1.1 and Figure 1.2 below. Figure 1.2 is the game with four players all connect to the game. The detailed interface explanations are in section 3.7. To support the above interactive network poker game, we used Servlet, XML, VRML, JavaScript, and DOM. The detailed description of each terminology follows:

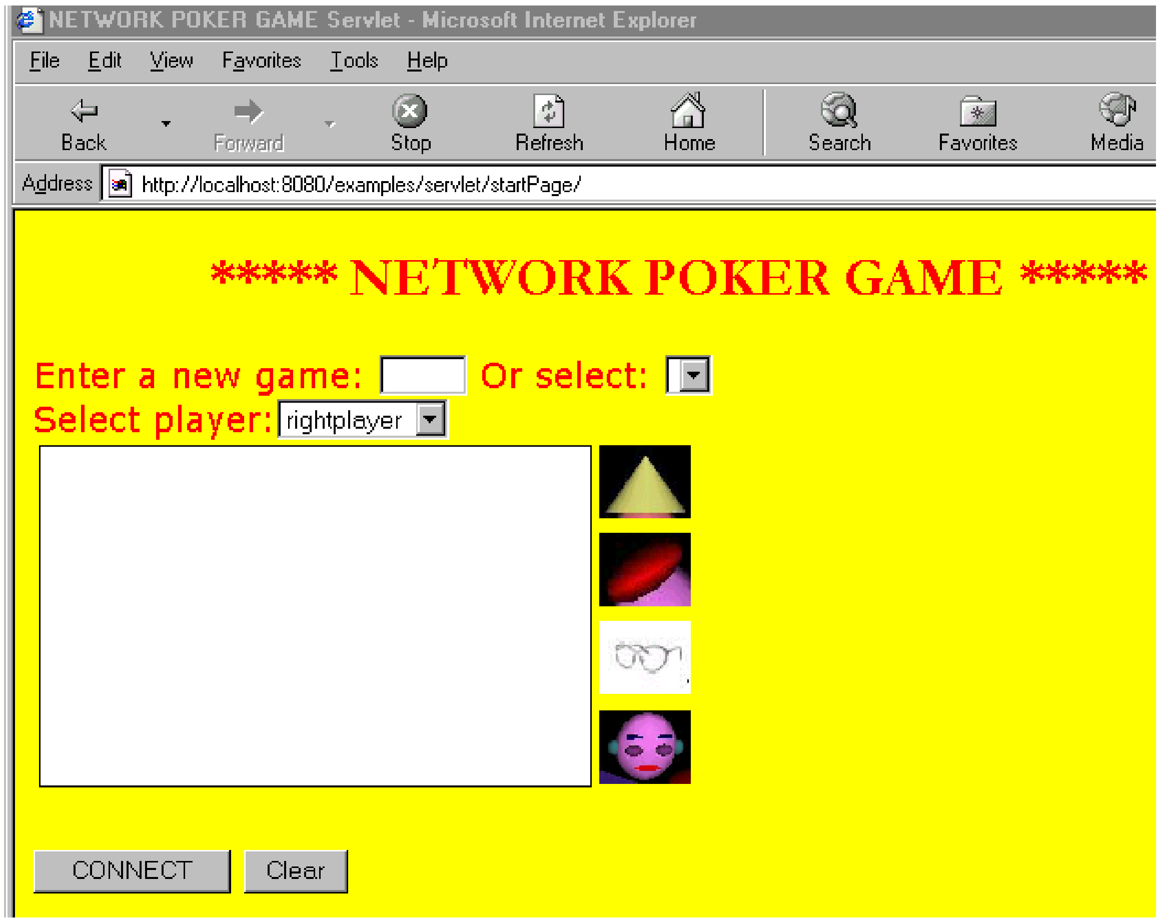
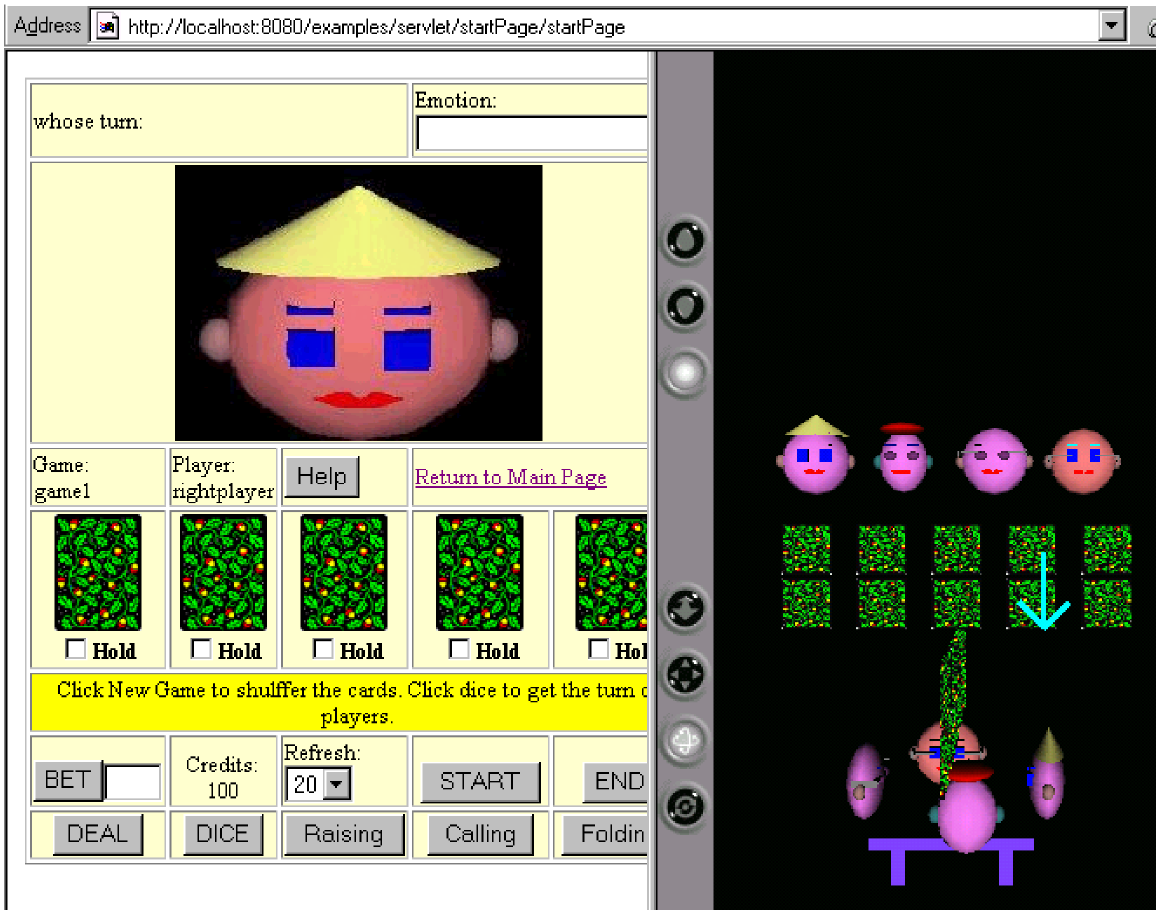

NETWORK POKER GAME Servlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media
Address http://localhost:8080/examples/servlet/startPage/
***** NETWORK POKER GAME *****
Enter a new game: Or select:
Select player: rightplayer
CONNECT Clear

**Figure 1.1:** Start page of the project


Address http://localhost:8080/examples/servlet/startPage/startPage

whose turn:

Emotion:

Game: game1

Player: rightplayer

Click New Game to shulffer the cards. Click dice to get the turn o

players.

Credits: 100

Refresh: 20

**Figure 1.2:** Page allows user to play game with VRML 3D facial emotion

## *Servlet*

A servlet is a server-side software component, written in Java, that dynamically extends the functionality of a server. Servlets provide a framework for creating applications that implement the request/response paradigm. For example, when a browser sends a request to the server, the server may forward the request to a servlet. At this point the servlet can process the request and construct an appropriate response (usually in HTML) that is returned to the client. In this project, we use Java Servlet to handle request/response between server and client.

***XML***

An XML document is a tag-based document used to describe specified emotion that each player has. The following sample is an XML input file of a player with left eye blink and right eyebrow up emotions.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE players SYSTEM "http://localhost:8080/examples/servlets/players.dtd">
<players>
    <player name="john">
        <eyeblink which="left"></eyeblink>
        <eyebrowup which="right"/>
    </player>
</players>
```

The DTD file specifies what emotions a 3D character supports, and it is used to validate whether the XML source file of a player is correct.  For all emotion availabilities, see section 4.1.

***VRML***

VRML (pronounced vermal) stands for the Virtual Reality Modeling Language, a system for describing 3D scenes on the web. The following example is a VRML application. This example creates a basic primitive shape box with two fields for describing an object. The fields are the appearance and geometry fields. They describe the look and shape of the object respectively. Appearance nodes describe Shape’s appearance. Inside an appearance node, there is a material field containing a Material node with default values to create a shaded white appearance (if no value specified.) Material nodes can specify diffuseColor, emissiveColor, shininess, and transparency, and so on to create shaded, white shapes. Shape geometry is built with geometry nodes. A geometry node field controls dimensions. For example, the Box geometry node builds a box; its field contains a size field that defines the width, height and depth of the box.

```
Shape{
        Appearance Appearances {
                Material Material { }
        }
        geometry Box {
                size 2.0 2.0 2.0
        }
}
```

The VRML co-ordinate system works as shown in the diagram below:

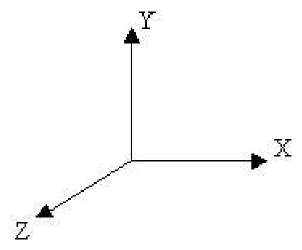


**Figure 1.3:** Co-ordinate Systems & Axes

The X-Axis is horizontal, Y is vertical, and Z is depth, coming out of the screen at you.

In VRML, all shapes are built at the center of the world by default. A transform operation can change the center of view. VRML has three types of transformations we can apply to objects. These are translation, rotation and scaling. All these can be used in a Transform node but doesn't have to have all three in it. We can just have a rotation, for instance. The syntax for this is shown in the example below.

```
Transform {
       translation  1   1   1
       rotation      0   1   0   0.78
       scale         2   1   2
       children [
                USE myBOX
        ]
}
```

A Transform node can have another nested node inside it as a child, which allows you to do sequences of transformations. Translation and Scale are very similar. Both take three arguments; x, y, and z values. Translation moves the center of the object these distances in the appropriate direction. Scale multiplies the size of the object by these values in the appropriate directions. Rotation takes four arguments. The first are three co-ordinates, which define the axis of rotation, and the last is the angle to rotate by, in radians.

The IndexedFaceSet node is a collection of faces that are defined manually so that we could build up our own objects of any shape. For example, suppose we want to define a triangle as below. We first need to define the four vertices, numbered 0-3 in the diagram. Then, we need to define the faces. To define the face, we use the vertices 0, 1, and 2.

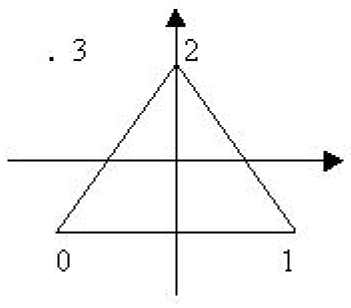


**Figure 1.4:** Points of IndexedFacedSet

The coord field contains a Coordinate node, which simply consists of a list of 3D points. These are the points that make up the IndexedFaceSet. The next field is coordIndex. This is the face list. To define a face, we enter the index of each point on it in this list.

The index of the point is its number in the coordinate list. If we are looking at the face, the index values should be entered in anticlockwise order for the face to be facing us, and not away from us. The example below defines a cube with four faces missing, which isn't solid.

```
geometry IndexedFaceSet {
   coord Coordinate {
      point [
          -0.5 -0.5 0.0, 0.5 -0.5 0.0
           0.0   0.5 0.0, -0.5 0.5 0.0
      ]
   }
   coordIndex [ 0, 1, 2]
}
```

In VRML, we can provide our own preferred viewpoints to select the entry point position or select favorite views for the viewer. We can name the views for a browser menu. Viewpoints specify a desired location, an orientation, and a camera field of view lens angle. An example of the Viewpoint node is as follows:

```
Viewpoint {
        description "Corner view"
        position 3.0 1.6 3.0
        orientation 0.0 1.0 0.0 0.611
}
```

VRML browsers provide menus to specify the appearance of your avatar. The NavigationInfo node controls the movement features of the avatar. The NavigationInfo node works in concert with the current Viewpoint node. The Viewpoint node describes how to view a world, and the NavigationInfo node describes how to move about in that world and at what speed rate. The value of the type field can be WALK, Fly, EXAMINE, or NONE. Most VRML browsers provide menu items to speed up or slow down the avatar’s movement, overriding your speed hint. An example of the NavigationInfo node is as follows:

```
NavigationInfo {
        type "WALK"
        speed 1.0
        headlight FALSE
        avatarSize [ 0.5, 1.6, 0.5 ]
},
```

VRML gives us the ability to create our own behaviors to make shapes move, rotate, scale, blink, and more. Almost every node can be a component in an animation circuit. Wired routes connect nodes together. For example: to spin a shape, just connect a node that sends rotation events to a Transform node's rotation field.

An event is a message sent between nodes. As well as a number of fields, most nodes contain events. There are two types, eventIn and eventOut. eventOuts are outgoing events, which generate information such as the changing of a value, or the time of a mouse click. eventIns are incoming events, which accept information from outside the node and do something with it. These are an eventIn and eventOut for the field that can be used to set its value and notify the outside world when it has been changed. If we use set_fieldname to set the value of the field, the node then generates a fieldname_changed event. If a field is not exposed, it cannot be changed by events, and the value in the file is the one used at all times.

In order to do useful things with events, we need to somehow wire them together. This wiring is known as a ROUTE. For example, to route the eyes changing color, we would route the events as follows:

```
ROUTE eyeblinkTouchSensor.isOver TO CLOCK.set_enabled
ROUTE CLOCK.fraction_changed TO COLOR_PATH.set_fraction
ROUTE COLOR_PATH.value_changed TO eyeColorRight.set_diffuseColor
```

In order to do route, we need to define a name for the node in our 3D face world. Once a node has a name, we can use that node again later in the file. For instance, we can specify the name eyeColorRight for a node or group of nodes that build right eye's color, which is used in route. We need to use DEF for each node that we route to or from. The syntax for defining a name is as follows:

```
DEF node-name node-type { … }
```

Sensors can be divided to two categories, those associated with geometry and those for sensing time. Sensors can respond to various conditions such as proximity to an avatar, mouse clicks, or mouse movement. In addition, cylinder, sphere and plane sensors move the geometry associated with them when they are active.

Interpolator nodes are very important for animation. They are used to change a certain value over time. Depending on what value we want to change, there are six different interpolator nodes provided in VRML and X3D.

We use an interpolator when we want to change a value over time. An interpolator takes timing signals from a TimeSensor or similar sensor, and performs a linear interpolation between sets of values called keyValues. For each keyValue there is a key, which is a fraction from 0 to 1. A timeSensor outputs fraction_changed events that can be routed to the set_fraction eventIn of an interpolator to set the correct point in the interpolation cycle.

In our project we used several basic primitive shapes such as box, cone, and sphere to create players' faces and cards. We also used sensor, translation, scaling, rotation, grouping, mapping texture, IndexedFaceSet, route, viewpoint, and NavigationInfo as we described above to make our facial emotional character agents more interesting and realistic. The detailed coding is explained in section 4.6.

### *JavaScript*

We used JavaScript to add interactivity to our network poker game.  A JavaScript is a program that is included on an HTML page. JavaScript allows us to create an active user interface. For example, in the first page of the project, a user can click an image to choose the type of player he or she wants to create. There are different areas on the image of the face on the playing game page that a user can click for sending emotion messages. The "HELP" button opens a new window; display the network poker game's rules and the information of network poker game. Those are all done with JavaScript, using map, onclick (), and onchange () etc. Detailed description of methods used in JavaScript such as opening VRML mime type is in section 4.5.

### *DOM (Document Object Model)*

In this project, we take player.xml as an input source document and translate it into 3D VRML to display a player's emotion scenes. During the translation process, we use DOM API (Document Object Model Application Programming Interface) to traverse a tree-structure XML source document so that we can get a parse structure of the input source document. We use this to determine what emotion availability the player has so that we can produce a correct 3D emotion. The details about how we use DOM API to access and manipulate a source XML file and VRML fields in the output document on the fly are described in section 4.4. Once we know what emotion the player has and how we can generate. The next step is to play network poker game.

This network poker game is a basic five-card draw poker, which allows four persons to play. When a new game starts, the deck is shuffled, and the starter of the round is randomly generated. Draw Poker involves each player being dealt a number of cards, followed by a betting round, and followed by a "draw". The draw allows each player to exchange a number of his or her cards for different cards from the deck that may better suit the player's hand. For example, a player who has three of five cards that are not doing anything to help his or her overall hand discards those three cards and is dealt three new cards from the deck (called "blind cards", because a player does not know what they will be). After dealing and betting, players can call, raise, and fold. Detailed network poker game rules are described in section 4.7. Player's facial emotional message can be sent to server to display corresponding 3D emotions based on players' cards.

# 2. Design

## 2.1 Data Flow Diagram

Below is a data flow diagram showing processes in this application along with inputs and outputs.

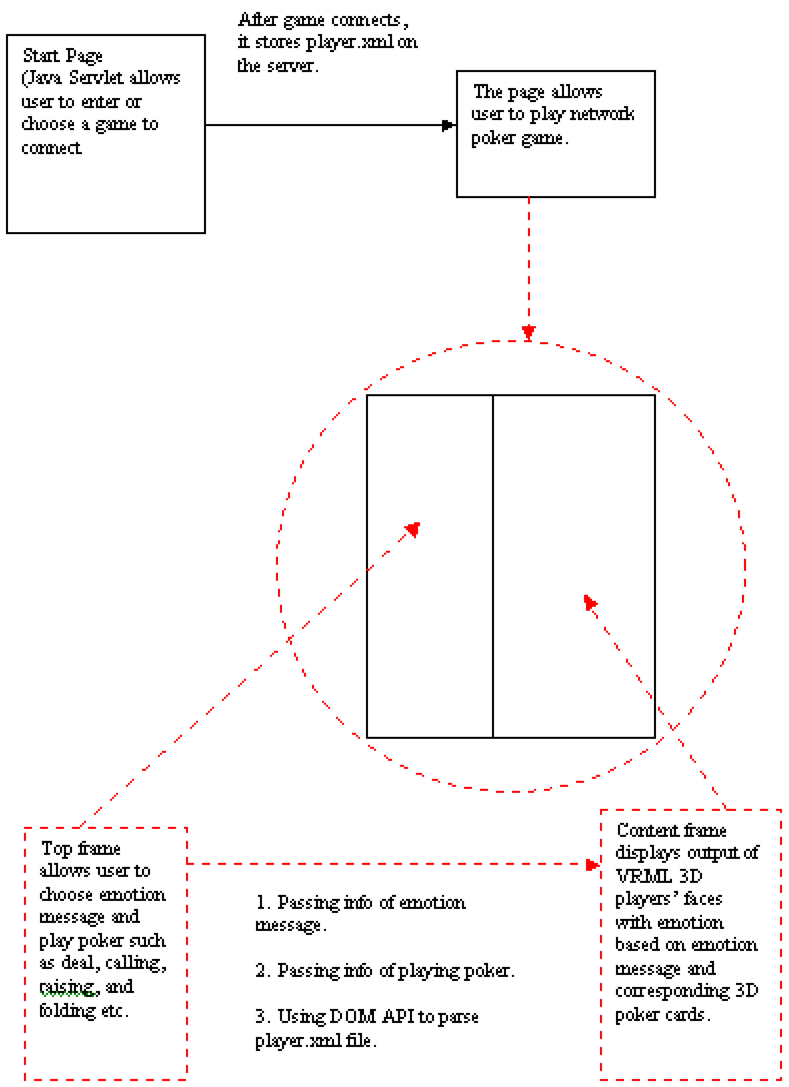

Start Page (Java Servlet allows user to enter or choose a game to connect
After game connects, it stores player.xml on the server.
The page allows user to play network poker game.
Top frame allows user to choose emotion message and play poker such as deal, calling, raising, and folding etc.
1. Passing info of emotion message.
2. Passing info of playing poker.
3. Using DOM API to parse player.xml file.
Content frame displays output of VRML 3D players' faces with emotion based on emotion message and corresponding 3D poker cards.

**Figure 2.1:** Data flow diagram of this application

## 2.2 Program Design

This application is developed using an objected-oriented approach. The startPage, gameServlet, and gameServletTwo classes use Hand, Card, Deck, Cash, Bet, Score, sortScore classes (see Figure 2.2).

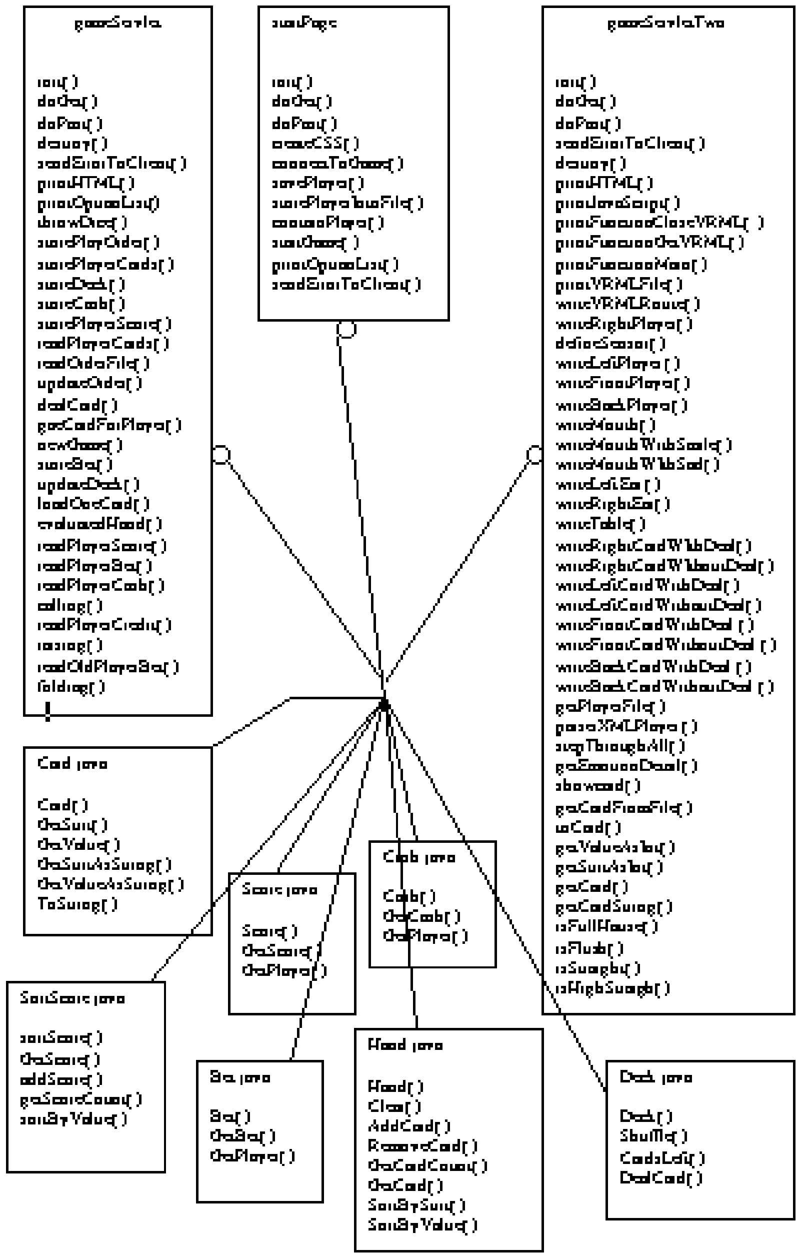


**Figure 2.2:** UML diagram of all classes.

# 3. Requirements

## 3.1 Browser

The ideal browser for us to display the output document is Internet Explorer 5.0 or later versions because it supports VRML – Cortona plug-in that is available at www.parallelgraphics.com/products/cortona. The Cortona plug-in is used for this project.

VRML can also be displayed without a plug-in. For example, it can be displayed with the Java and Java3D applications such as blaxxun3D and Xj3D etc. The detailed information about applications, which display VRML, is in Table 3.1. It contains the information of software type, operating system, and web browser.

| **Software (Disclaimer)** | **Software Type** | **Operating System** | **Web Browser** |
|---|---|---|---|
| Cosmo Player | Plug-in | Windows & Mac | IE & Netscape |
| Cortona | Plug-in | Windows & Mac | IE & Netscape |
| BS Contact VRML | Plug-in | Windows | IE & Netscape |
| Flux | Plug-in | Windows | IE |
| Octagon Player | Plug-in | Windows | IE |
| Vcom3D Venues | Plug-in | Windows | IE |
| Blaxxun Contact | Plug-in | Windows | IE & Netscape |
| OpenWorlds | Plug-in & program | Windows | IE |
| FreeWRL | Plug-in& program | Linux & Mac | Netscape |
| OpenVRML-Lookat | Plug-in & program | Linux | Netscape |
| BS Contact J | Applet | Windows & Linux & Mac | IE & Netscape |
| Shout3D | Applet | Windows & Linux & Mac | IE & Netscape |
| Cortona Jet | Applet | Windows & Linux & Mac | IE & Netscape |
| Blaxxun 3D | Applet | Windows & Linux & Mac | IE & Netscape |
| FastScript3D | Applet | Windows & Linux | IE & Netscape |
| Xj3D | program | Windows & Linux | |

**Table 3.1:** VRML & X3D Plug-in

## 3.2 Software

All of the classes and interfaces required to create and execute servlets are contained within two packages—javax.servlet and javax.servlet.http. The java Servlet API is a collection of classes and interfaces, and it is part of Sun's Java Servlet Development Kit (JSDK). In this project, we use J2sdk1.4.1, which can be found at http://java.sun.com/j2se/1.4.1/install-windows-64.html. We choose Tomcat4.1, which is a free, open-source implementation of Java Servlet and JavaServer Pages technologies developed under the Jakarta project at the Apache Software Foundation as our web server, and it can be found at http://java.sun.com/products/jsp/tomcat/.

The Xerces2 Java Parser 2.2.1 supports the standard DOM API, and it is available at http://xml.apache.org/xerces2-j/index.html.

## 3.3 Running Servlets

Today, there are many options available for running servlets such as JSDK Servlet Runner, Java™ Web Server, JRun™, and ServletExec™ etc. With Servlet Runner, servlets are invoked using the following URL format:

/servlet/<servlet_name>[/path_info>][?<query_string>]

The information displayed in brackets is optional. The <servlet_name> portion of the URL represents the servlet alias or servlet class name. For example, the network poker game's start page servlet can be invoked using the following URL:

http://localhost:8080/examples/servlet/startPage

## 3.4 XML file

A source player.xml file is validated with the player.dtd file after it is generated onto the server side directory on the fly. The translator will not check any syntax errors of a source document. Nothing will be displayed on the output page if there are syntax errors in a source file. In this project, it is a programmer's responsibility to write the correct player.xml on the server side directory since the player.xml is generated on the fly. A well-formed XML must satisfy these constraints:

1. All tags must have a corresponding ending tag
2. No overlapping tags

## 3.5 DTD file

We create our own player.dtd file to express human facial behaviors according the studying of HumanML and Multimodal Presentation Markup. Besides the validation of a source document, another advantage of having a DTD file is that we can define emotion availabilities as we need and a default value to any attribute in a DTD file. The default value is automatically given to the translator through an XML parser if that attribute is not specified in the source document. We used xmlspy 5.0, which is the industry-standard XML development environment, to check whether player.dtd and player.xml are well formed and to validate the XML documents. The xmlspy 5.0 can be found at http://www.xmlspy.com/.

(The player.dtd and sample player.xml files are included in *Appendix A*.)

## 3.6 XML Tags Supported in This Project

Not all tags that we defined in our DTD are displayable in our game. The table below shows the XML tags and attributes that we support in this project.

| Tag | Attributes | Values |
|---|---|---|
| players | - | - |
| player | name | - |
| eyeblink | which | left \| right \| both |
| eyebrowup | which | left \| right \| both |
| eyebrowdown | which | left \| right \| both |

**Table 3.2:** XML tags and attributes supported in the project

## 3.7 User Interface

The starting page of the project allows a user to enter a new game name (a game never connected to before) or choose an existing game name (if the game has already been created) from a drop down list. Use can choose which player to connect to the game from a drop down list such as rightplayer, leftplayer, frontplayer, and backplayer. It also allows users to choose different images to build up a player's face. For example, if user clicks on a cone image, then the player will have a cone head. If user clicks on the glasses image, then the player will have a pair of glasses on the face. If a player wears a pair of glasses, she or he will not have eye blink emotion. Next a user clicks the **CONNECT** button to connect to the game he/she selected (See Figure 1.1 in section 1.2). This brings the user to a network poker game page (See Figure 1.2 in section 1.2). It contains two frames: a left frame and a content frame that is on the right side. The left frame allows the user to select an emotion message (See Figure 3.1) and play a poker game. It also passes the emotion message to the content frame to display a 3D VRML of the players' face with their poker cards. Users can go back to the start page by clicking **Return to Main Page.** The 'Help' button opens a new window and displays the information of the network poker game such as game rules (the steps for how to play this network poker game) and examples of the poker hand.

In the left frame, each time the user chooses either emotion or changes the refresh time parameter as well as presses a button; it will send the request to the right frame to display the corresponding 3D character.

We use image mapping to implement users sending emotion messages. Figure 3.1 shows the image map area that user can click. When 'User' clicks <eyebrowup left> area, the response sends the left eyebrow up emotion message to display 3D eyebrow up emotion. Clicking <eyebrowdown left> area sends left eyebrow down emotion message to display 3D left eyebrow down emotion. Clicking <eyebrowup both> sends both eyebrows up emotion message. Clicking <eyebrowdown both> sends both eyebrows down emotion message. Clicking <eyebrowup right> sends right eyebrow up emotion message. Clicking <eyebrowdown right> sends right eyebrow down emotion message. Clicking <eyeblink left> sends left eye blink emotion message. Clicking <eyeblink both> sends both eyes blink emotion message. Clicking <eyeblink right> sends right eye blink emotion message. The emotion message is displayed in the emotion box (See Figure 1.2).

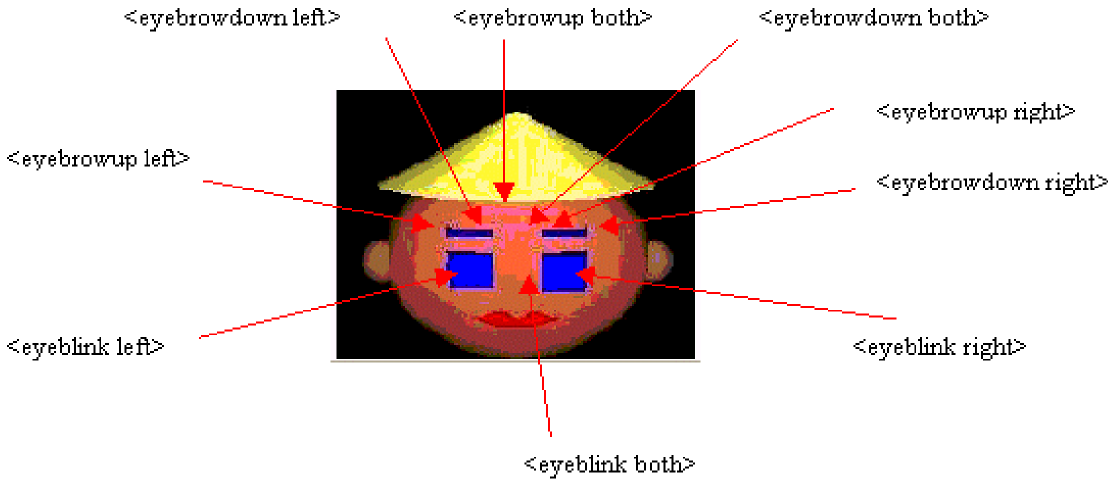


**Figure 3.1:** Image Mapping Areas

Another emotion that we coded in this project is the smile, which is the default emotion. This smile emotion displays only when a player has a good hand such as flush, straight, royal flush, and straight flush etc. It is not selectable by the user.

In the beginning, if there is one player connecting to the game, Figure 1.2 page will show one player and his or her hand in 3D graphic. If there are two players connecting to the game, Figure 1.2 page will show two players and their hand in 3D graphic. It is user's responsibility to wait until four players are all connecting to the game, and then the game can be started to play.

### 3.8 Servlet Requirements

1. The servlet can enter a game name and store this game into a drop-down list. It also can choose a game from the drop-down list. In this manner, the user can dynamically add a game. No remove game operation is provided. (A second phase of this project might add the remove operation to allow a user to remove a game).

2. After selecting a game of interest to connect, the servlet should allow the user to add a player for the game. The player for each game should be stored in a separate text file such as gameX.txt. A simple append operation can be used to add a player to the text file. No update, delete, or sorting operation is required. (Again, a second phase of this project might allow a user to update, edit, and delete players.)
3. After connecting to a game, the player.xml file is written on the fly onto server side.
4. The game page must have a link back to the main page.
5. All players' information such as cards, bet, hand score, and credit etc are all stored in files on server side directory.
6. The servlet should retrieve the location of the text files described in requirement 2 and 5 from an initialization parameter set on the server. In this way, the location of these files can be easily changed without recompiling the servlet.
7. All data that are created in the playing game are saved in different files. (Again, a second phase of this project might allow the removal of all the data files after a game is over.) When a player is doing an action such as dealing, calling and folding etc, the corresponding files need to be locked, and the lock is released after the action is done.

# 4. Implementation

## 4.1 Emotions and Gesture Avatar DTD

DTD (Document Type Definitions): encoding character avatars to which facial animation, facial emotion, and gesture information can be readily applied.

DTD uses the elements below to facilitate the direction of a Virtual Human interacting with a user or standalone application. These elements are:

- players
- player
- facial animation elements
- emotion elements
- gesture elements

The elements of DTD are divided into four levels, where the top-level only constitutes one element tag <players>. At the second level the element is <player> that controls each player's information and his/her emotions and gestures. It must have at least one player. The third level element controls the emotions and the gestures, facial animation, emotion and gesture. Detailed description of each level element is in player.dtd, which is in Appendix A. The element structure of DTD is shown in Figure 4.1. The arrow indicates that the element on the upper level constitutes of the element on the low level.

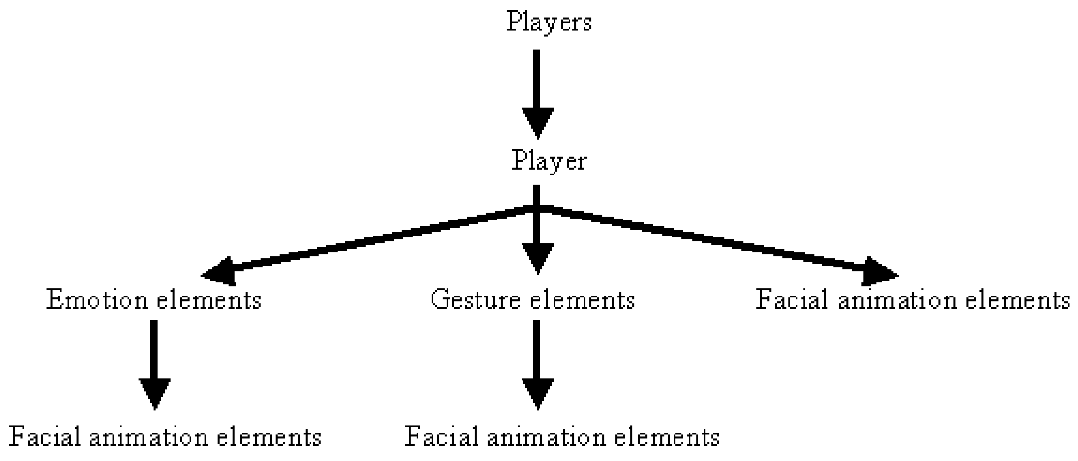

Players
Player
Emotion elements
Gesture elements
Facial animation elements
Facial animation elements
Facial animation elements

**Figure 4.1:** The element structure of DTD

As with XML elements, all elements of DTD are case sensitive; therefore all elements must appear in lower case and will otherwise cause a fatal error. When creating a DTD document, the first line must contain an XML declaration followed by a DTD specification. The input has to follow the rules of the structure of the DTD that is specified in player's DTD, otherwise, the user will get an error message when he or she tries to open the XML file through the IE.

Example:

```
<?xml version="1.0"  standalone="yes"?>
<!DOCTYPE players SYSTEM "player.dtd">
<?xml-stylesheet type="text/xsl" href="player.xsl"?>
......
```

### 4.1.1 Top level element

The elements at the top level control the structure of the language as well as specify the game.

**<players>**

Description: Root element that encapsulates all other elements.
Attributes:

| Name | Description | Values | Default |
|---|---|---|---|
| xml:lang | Indicates the language on the enclosing element | A language code. | optional |

Properties: Can only occur once.
Can contain <player> element.
Example:

```
<players>
        …
</players>
```

### 4.1.2 Second level element

**<player>**

Description: Specifies the player.
Attributes:

| Name | Description | Values | Default |
|---|---|---|---|
| name | Indicates the player's name. | String | Required |

|  |  |  |  |
|---|---|---|---|
| Position | Indicates the player's position | integer | Required |
| disemotion | Indicates the starting emotion the player has. (It might be the special emotion the player has and affect all emotion the player will apply.) | string | Defaultemotion |

Properties: Can only occur once.
Can contain <game> element.
Example:

```
<players>
        <player name="Jone" position="1" disemotion="defaulteomotion">
                <look-up/>
                …..
        </player>
        <player name="Alex" position="2" disemotion="smile">
                <look-up/>
                …..
        </player>
</players>
```

### 4.1.3 Third level element

#### 4.1.3.1 Emotion

The elements in emotion will affect the emotion shown by the Virtual player. This level element can contain the sub element of facial animation elements (which is the fifth level elements, see example).

##### 4.1.3.1.1 Emotion default attributes:

| Name | Description | Values | Default |
|---|---|---|---|
| duration | Specifies the time span in seconds or milliseconds that the emotion will persist in the player. | #s or #ms | required |
| intensity | Specifies the intensity of that particular emotion, either by a descriptive value or by a numeric value (Medium represents a numeric value equal to fifty). | A numeric value<br>• Low<br>• medium<br>• high | medium |

Each element has at least two attributes associated with it.

4.1.3.1.2 Emotion elements:

<afraid>, <angry>, <confused>, <dazed>, <disgusted>, <happy>, <smile>, <neutral>, <sad>, <surprised>, <concentrate>, <default-emotion >

Example:

```
<sad>
	<look-down wait="750ms"/><look-up/>
</sad>
<smile intensity="50" duration="5000/>
```

### 4.1.3.2 Gesture

The elements in gesture will accommodate well-known human gesture. This level element can contain the sub element of facial animation elements (which is the fifth level elements, see example).

4.1.3.2.1 Gesture default attributes

Each element has at least two attributes associated with it (see emotion default attributes).

<agree>, <disagree> attribute:

| Name | Description | Values | Default |
|---|---|---|---|
| repeat | Time the action repeated | integer | 1 |

4.1.3.2.2 Gesture elements

<agree>, <disagree>

Example:

```
<agree>
	<look-up wait="750ms"/>
</agree>
<disagree intensity="30" duration="1200"/>
```

### 4.1.3.3 facial animation

The elements in facialanimation affect the facial animation performed by the Virtual player. These elements will only make changes to the face.

#### 4.1.3.3.1 facialanimation default attributes

Each element has at least two attributes associated with it (see emotion default attributes).

<eyebrow-up>, <eyebrow-down> attribute:

| Name | Description | Values | Default |
|---|---|---|---|
| which | which eyebrow to move | Both, right, left | both |

<eye-blink>, <wink> attribute:

| Name | Description | Values | Default |
|---|---|---|---|
| repeat | Time the action repeated | integer | 1 |

#### 4.1.3.3.2 facialanimation elements

<look-left>, <look-right>, <look-up>, <look-down>,
<eyes-left>, <eyes-right>, <eyes-up>, <eyes-down>,
<head-left>, <head-right>, <head-up>, <head-down>,
<head-roll-left>, <head-roll-right>,
<eyebrow-up>, <eyebrow-down>,
<eye-blink>, <wink>, <open-jaw>, <close-jaw>

Example:

```
<look-down wait="750ms"/><look-up/>
<eyebrow_up which="left" duration="1000"/>
<head_left intensity="40" duration="1000"/>
```

## 4.2 JAVA SERVLET

The web pages of Network Poker Game are all written in Java Servlet, which allow interaction with users and dynamically write HTML.

The start page (See Figure 3.1 in section 3) of the network poker game servlet allows a user to select a game of interest from a drop-down list, or enter a new game in the input box. The drop-down list is populated with all existing games, which are stored in a

Vector. The user can choose to build a player's face with specific emotion availabilities XML file. The player.xml is written on the fly and stored on the server side directory. The CONNECT button allows the user to connect to the game page. After connecting, all the information of players and games is stored into different files on the server, e.g., the players' names are stored in players.txt, and current games are stored in gameName.txt etc. The CLEAR button clears the text in the input box when the user enters anything wrong.

Let's take a look at the code that generates this screen. First, the servlet must retrieve the directory where all information of players and games is stored from an initialization parameter. This operation is performed in the servlet's init () method shown as follows.

```
//instance variables
String filePath;  //Store path for all files such as players.txt, and gameName.txt etc.
/**
 * init method is called when servlet is first loaded.
 * It reads from an initialization parameter the directory
 * where all files are stored.
 */
public void init(ServletConfig config) throws ServletException
{
        super.init(config); //pass ServletConfig to parent

        //get path to support files
        filePath = config.getInitParameter("FilePath");

        if(filePath == null)
        {
                //default directory if no FilePath init parameter exists
                filePath = "D:/Program Files/Apache Group/Tomcat
4.1/webapps/examples/WEB-INF/classes/";
        }
}
```

The variable filePath is declared as an instance variable outside of any method. In this way, all requests serviced by this servlet will have access to its value. Since the value of filePath is set in the init () method and does not change, the use of an instance variable is safe and will not lead to errors and inconsistent results.

Next, doGet () method is invoked in response to any HTTP GET request. The method begins by setting the content type of our response to HTML and proceeds to get a handle to the output stream. Because we only return ASCII text, a PrintWriter object is declared.

Then we use PrintWriter out to send HTML to the client. The HTML generates the main page as we see in Figure 3.1.

```
/**
 * doGet method is called in response to any GET request.
 * Returns an HTML form that allows the user to enter a new game
 * or choose a existing game and choose different image to build player's face.
 */
Public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException
{
        // Mime type to return is HTML
        response.setContentType("text/html");

        // get a handle to the output stream
        PrintWriter out = response.getWriter();

        try
        {
                // generate HTML form requesting
                out.println("<HTML><HEAD>");
                out.println("<TITLE>NETWORK POKER GAME Servlet</TITLE>");
                printJavaScript(out);  // Print out Javascript part of HTML
                out.println("</HEAD>");
                out.println("<BODY>");
                createCSS(out);  // Cascading Style Sheets
                out.println("<center><H1>**   NETWORK POKER GAME **
</H1>");
                out.println("</center>");
                out.println("<FORM name=myForm METHOD=\"POST\"
ACTION=\"startPage\">");
                out.println("<input type=hidden name=num1 value=0>");
                //out.println("<input type=text name=num2 value=0>");

                // Create HTML input box for user to enter new game name
```

```
                    out.println("<P>Enter a new game: <INPUT TYPE=\"TEXT\"
NAME=\"newName\" " + "SIZE=\"5\">");

                    // Create HTML drop down list to allow user selecting existing game name
                    out.println("Or select:");
                    out.println("<select NAME=\"gameName\" SIZE=1>");
                    printOptionList(out);
                    out.println("</select>");
                    out.println("<br>");

                    //Allowing user choose the player image
                    out.println("<table>");
                    out.println("<tr>");
                    out.println("<td rowspan=\"4\" valign=\"center\">");
                    out.println("<IMG name=\"myIMG\" SRC=\" " + filePath +
"/images/blank.jpg\" BORDER=1 WIDTH=300 HEIGHT=185>");
                    out.println("</td>");

                    out.println("<td  valign=\"center\">");
                    out.println("<IMG SRC=\" " + filePath + "/images/hatCone.jpg\"
BORDER=0 WIDTH=50 HEIGHT=40 onClick=\"changeToCone();\">");
                    out.println("</td>");
                    out.println("</tr>");

                    out.println("<tr>");
                    out.println("<td valign=\"center\">");
                    out.println("<IMG SRC=\" " + filePath + "/images/hatSphere.jpg\"
BORDER=0 WIDTH=50 HEIGHT=40 onClick=\"changeToSphere();\">");
                    out.println("</td>");
                    out.println("</tr>");

                    out.println("<tr>");
                    out.println("<td valign=\"center\">");
                    out.println("<IMG SRC=\" " + filePath + "/images/eyeGlass.jpg\"
BORDER=0 WIDTH=50 HEIGHT=40 onClick=\"changeToGlass();\">");
                    out.println("</td>");
                    out.println("</tr>");

                    out.println("<tr>");
                    out.println("<td valign=\"center\">");
                    out.println("<IMG SRC=\" " + filePath + "/images/normalPlayer.jpg\"
BORDER=0 WIDTH=50 HEIGHT=40 onClick=\"changeToNormal();\">");
                    out.println("</td>");
                    out.println("</tr>");
                    out.println("</table>");

                    // Create HTML submit BUTTON
                    out.println("<BR/><input TYPE=\"SUBMIT\" NAME=\"connectGame\"
VALUE=\" CONNECT \">" +
                                        "      " + "<INPUT TYPE=\"RESET\" VALUE=\" Clear
\"></p>");
                    out.println("</form>");
                    out.println("</BODY></HTML>");
          …. More code here
}
```

After the user clicks the CONNECT button on the start page, the doPost () is called in response to an HTTP POST request. We use output stream out to transmit HTML back to the client.

```
/**
 * doPost method is called in response to any POST request.
 * This method determines if it was called in response to the user
 * connecting to a game accordingly.
 */
public void doPost(HttpServletRequest request, HttpServletResponse response)
{
          String currentGame = null;
          response.setContentType("text/html");  // html output
          PrintWriter out = null;
          // get request
          String name = request.getParameter("newName");   //user entered new game
name
          String game = request.getParameter("gameName");  //existing game name
          String image = request.getParameter("num1");     //image that user chooses
          if( name != null && name.length() > 0 )
          {
                    existingGameName.addElement(name);
                    currentGame = name;
          }
          else   // name == null
          {
                    currentGame = game;
          }

          //More code here ….
          saveName(currentGame, "gameName.txt", out);
          saveName(currentPlayer, "currentPlayer.txt", out);

          //More code here
          try
          {
                out = response.getWriter();  // get handle to output stream
                if(request.getParameter("connectGame") != null)
                {
                        connectToGame(out);
                }
        }
        catch (Exception e)
        {
                // send stack trace bace to client
                sendErrorToClient(out, e);
        }
        out.close();
}
```

The connectToGame () method writes HTML code that generates two frames for the real game page.

```
public void connectToGame(PrintWriter out)
{
        //More code here …
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Main frameset of HTML </title>");
        out.println("</head>");
        out.println("<frameset cols=\"50%, 50%\">");
        out.println("<frame src=\"http://localhost:8080/examples/servlet/gameServlet\"
name=\"leftTop\"scrolling=\"auto\"> ");
        out.println("<frame
src=\"http://localhost:8080/examples/servlet/gameServletTwo\" name=\"content\"
scrolling=\"auto\"> ");
        out.println("</frameset>");
        out.println("</html>");
}
```

In catch block of doGet () and doPost (), the sendErrorToClient () method is called to return an error message, which all exceptions are caught and sent to the client.

```
/**
 * Return stack trace to client. Useful for debugging
 * @param out Client output stream
 * @param e Exception
 */
private void sendErrorToClient(PrintWriter out, Exception e)
{
        //send stack trace back to client and to standard out
```

```
            StringWriter stringError = new StringWriter();
            PrintWriter printError = new PrintWriter(stringError);
            e.printStackTrace(printError);
            String stackTrace = stringError.toString();

            //send error message to client
            out.println("<HTML><TITLE>Error</TITLE></HTML><BODY>");
            out.println("<H1>Servlet Error</H1><H4>Error></H4>" + e +
                        "<H4>Stack Trace</H4>" + stackTrace + "</BODY></HTML>");

            //print stack trace to standard out
             System.out.println("Servlet Error: " + stackTrace);
}
```

The game page (See Figure 3.2) contains two frames, left frame and right frames. The gameServlet.java generates the content of left frame, and the gameServletTwo.java generates the content of right frame. Similar to startPage.java, they are all written in java servlet by using basic doGet () and doPost () methods and some other functions to make network poker game and emotion in 3D work. The detailed coding of startPage.java, gameServlet.java and gameServletTwo.java is in *Appendix A*.

## 4.3 Using the RequestDispatcher Object

To send emotion messages from one servlet to another servlet, we use Request Dispatching. Request Dispatching enables us to write applications where one servlet processes a request partially and then invokes another servlet, or HTML page, to do further processing. This functionality is provided in the Servlet API in the form of the *javax.servlet.RequestDispatcher* interface. In this project, we forward a client's emotion message to the server and from the server to another servlet.

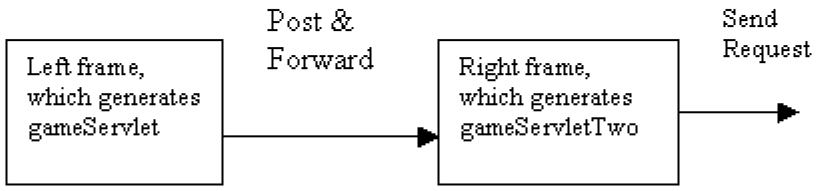

Left frame, which generates gameServlet
Post & Forward
Right frame, which generates gameServletTwo
Send Request

**Figure 4-2** An example of how Request Dispatching may be used

The information collected through the HTML form, which is generated by gameServlet, is posted to the gameServlet. Information collected through the form is passed on to the gameServlet through the request object. The gameServlet sends the information to gameServletOne. The gameServletOne sends the information to gameServletTwo and sends a response to the client browser. GameServletTwo gets the emotion message and generates a corresponding response to the client browser.

In gameServlet.java, we forward the request to gameServletOne.java. The following is the sample code.

```
RequestDispatcher rd = request.getRequestDispatcher("/servlet/gameServletOne");
rd.forward(request, response);
```

The forward() method of the RequestDispatcher object allows us to programmatically forward a request to another servlet (or another server resource). Because forwarding passes all control to the new resource, we do not disrupt its communication with the client by retrieving a handle to the output stream. The forward() method of the RequestDispatcher object passes the current request to the servlet designated by the specified URL ( /servlet/gameServlet). Unlike the include() method, control will never retrun to the servlet that forwards the request. The forward() method passes all control to the delegate servlet which will generate the response.

The include() method of the RequestDispatcher object allows us to programmatically include content generated by another servlet (or another server resource) within the body of the calling servlet's response. The included servlet cannot change the status of the response. Using the include(…) method of the RequestDispatcher, we can include the output of other Servlets at any point in the including serlet. For example gameServletOne can include gameServletTwo's output into own with the following code.

```
RequestDispatcher rd = request.getRequestDispatcher("/servlet/gameServletTwo");
rd.include(request, response);
```

## 4.4 Using DOM API to Parse XML File

In order to know what kind of emotion availabilities each player has, we have to explore each player.xml file for the game. XML documents have a hierarchy of informational units called nodes; DOM is a way of describing those nodes and the relationships between them. We use a tree-based DOM API to parse player XML input file to know the emotion availabilities the player has.

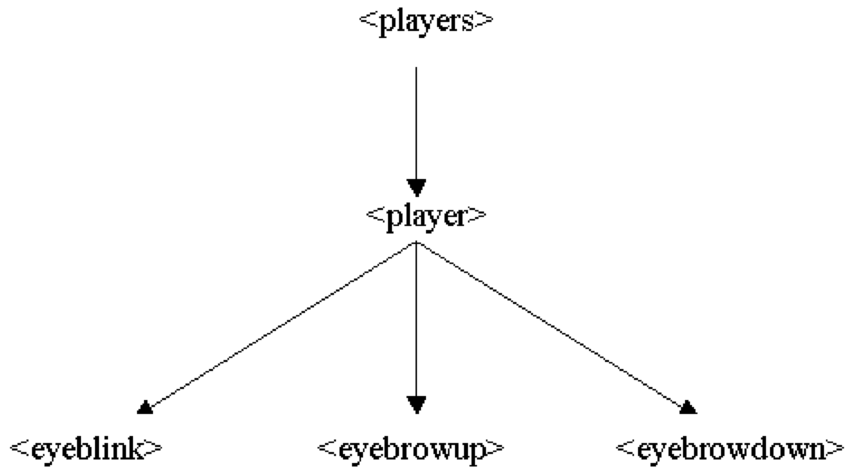


**Figure 4-3:** Tree structure of XML tags supported in the project

The following code first parses the XML file and creates a tree structure in the memory. A validating parser checks the XML file against the rules imposed by the DTD. A non-validating parser doesn't validate the XML file against a DTD. Both validating and non-validating parser check for the well formed of the XML document.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
dbf.setValidating(true);  // parser can be set to validate the document
doc = db.parse(docFile);
```

Once the Dom tree is constructed the stepThroughAll() method is called which traverses the tree and checks the content of each element node to see whether the player has the specific emotion attributes.

```
//Get the root element

Element root = doc.getDocumentElement();
NodeList playerList = root.getElementsByTagName("player");
//Loop through each player for xml file
for(int i=0; i < playerList.getLength(); i++)
{
        Element thisPlayer = (Element)playerList.item(i);
        // get the player infomation
        String playerName = thisPlayer.getAttribute("name");
        out.println("<BR>The player name is: " + playerName +".");
        out.println("<BR>Please load " + playerName + ".VRML file");
}
stepThroughAll(root, out, root);
```

## 4.5 JavaScript

In order to display 3D VRML, we use JavaScript to force visualization in a frame by opening the frame document with a VRML mime type association, and then having the JavaScript function write the appropriate VRML code:

```
out.println("<script language=\"javascript\" type=\"text/javascript\"> ");
out.println("<!-- Hide script from old browsers ");
out.println("function main () { ");
out.println("var doc = parent.content.document; ");
out.println("var doc = getVrmlDocument(parent.content.document); ");
out.println(" doc.open(\"model/vrml\"); ");
out.println(" doc.writeln(\"#VRML V2.0 utf8     \"); ");
//More code here is deleted
out.println(" doc.writeln(\"}                    \"); ");
out.println(" doc.close(); ");
out.println("}");

out.println(" function closeVrmlDocument() { ");
out.println(" var d = this.doc; ");
out.println(" d.open(\"text/html\"); ");
out.println(" d.write('<body leftmargin=\"0\" topmargin=\"0\" scroll=\"no\">'); ");
out.println(" d.write('<object id=\"Cortona\" width=\"100%\" height=\"100%\"
classid=\"clsid:86A88967-7A20-11D2-8EDA-00600818EDB1\">'); ");
out.println(" d.write('</object></body>'); ");
out.println(" d.close(); ");
out.println(" var e = d.all[\"Cortona\"].Engine; ");
out.println(" e.RootNodes.Add(e.CreateVrmlFromString(this.syntax)); ");
out.println(" } ");

out.println(" function getVrmlDocument(doc) { ");
out.println(" var isIE = navigator.appVersion.indexOf(\"MSIE\") != -1; ");
out.println(" if (!isIE)  ");
out.println(" return doc;  ");
out.println(" var adapter = new Object(); ");
out.println(" adapter.open = new Function(\"mimetype\", \"return;\"); ");
out.println(" adapter.write = new Function(\"s\", \"this.syntax += s;\"); ");
out.println(" adapter.writeln = new Function(\"s\", \"this.syntax += s; this.syntax
+='\\\\n';\");  ");
out.println(" adapter.close = closeVrmlDocument; ");
out.println(" adapter.doc = doc; ");
out.println(" adapter.syntax = \"\"; ");
out.println(" return adapter; ");
out.println(" } ");

out.println("// End hiding script from old browsers --> ");
out.println("</script> ");
```

## 4.6 VRML

We build and group predefined shapes to make a player's face and poker cards. We use translation, rotation, and scale to manipulate the primitive shape, as we need to make the facial emotional character agents more realistic.

```
String s = “Group {  children  [ Transform { translation 1.6 –1.1 0.0  scale 0.5 0.5 0.5  ” +
            “ rotation 0.0 1.0 0.0 –1.5708  children [DEF sqFacePlayer1 Transform ” +
            “{ scale 1.5 1.5 0.75  translation 0.0 0.0 –0.4 ” +  “ children [ Shape { ” +
            “appearance Appearance { material Material { diffuseColor 1 0.5 1 } ” +
            “ } geometry Sphere { radius 1.0 } } ] }    ”;
out.println(" doc.writeln(\" “ +  s + “   \"); ");
```

By using IndexedFaceSet, we build the mouth with points so that we can make player's smile with mouth's corner up.

```
String s = “ geometry IndexedFaceSet { “ +
            “  coordIndex [ 0 1 2 3 4 5 ]     “ +
            “  convex FALSE  “ +
            “  coord DEF COORD Coordinate {    “ +
            “  point [ 1 0.5 0, 0.5 0.2 0, -0.5 0.2 0, -1 0.5 0, -0.5 -0.2 0, 0.5 -0.2 0 ]   “ +
            “ } } } ] }    “;
out.println(" doc.writeln(\" “ + s + “  \"); ");
```

VRML enables one to take a picture of anything in the real world and paint, or map, it on any shape in one's VRML world. This technique is texture mapping. We use texture mapping for cards. For player's card, we used Mapping Textures technique to have a card's image on top of the box.

```
String s = “ Shape {  appearance Appearance { texture ImageTexture {   “;
out.println(" doc.writeln(\" “ + s + “  \"); ");
out.println(" doc.writeln('            url [ \" " + imgPath + "cardback.gif\" }} ] '); ");
String g = “ geometry Box { size 1 1 0.05      }}]}  “;
out.println(" doc.writeln(\" “ + g + “  \"); ");
```

We use route and sensor to generate emotion such as eye blink, eyebrow up, and eyebrow down etc.

```
String  rs = “DEF eyeblinkTouchSensor TouchSensor { } ” +
“ROUTE COLOR_PATH.value_changed TO eyeColorRightPlayer1.set_diffuseColor  ”;
out.println(" doc.writeln(\" “ + rs + “  \"); ");
```

It is often convenient to set up a predefined camera viewpoint in a world. Each time that world’s VRML file is loaded, the browser automatically positions the viewer at that predefined viewpoint position. From there, the viewer can move around in the world and view the world from additional positions and orientations. We use viewpoint and NavigationInfo to create different views of the poker game.

```
String v1 = “Viewpoint { description \”forward view\” position 2 2 16
                orientation 0.0 1.0 0.0 0.0 fieldOfView 0.7 }”;
String v2 = “Viewpoint { description \”90 degree Field-of-View from Corner \”
                position 17 1 3 orientation 0.0 1.0 0.0 1.5708 fieldOfView 0.7 }”;
out.println(" doc.writeln(\" “ + v1 + v2 + “  \"); ");
```

## 4.7 Poker Game (Five Card Draw Poker)

**Number of Players:**

In this network poker game project, we design to have four players. It can only play when there are four players connecting to a game.

**Cards:**

Cards in this project contain the entire deck, without jokers. Ace sometimes is high and sometimes is low. If ace is high, then two is low.

**Basic Rules:**

The network poker game uses a standard pack of 52 playing cards. The card ranking is as follows Ace (This may be the highest card depending on the variations we are using, but it is usually the lowest), King, Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3, 2, Ace (the lowest).

There are four suits (spades, hearts, diamonds and clubs). No suit is higher than another (Suits do not break ties). All poker hands contain five cards, and the highest hand wins. In this project, we compute the score of hand by using the following formula.

Score of Hand = (Value of Card in a Hand – 1) * 4 + ranking of the Hand

The formula before guarantees the correct ranking of the hand (see below for detail.) To distinguish different hands, we assign each hand a value. For example, high card is 0, a pair is 500, two pairs is 1000, three of a kind is 1500, straight is 2000, flush is 2500, full house is 3000, four of a kind is 3500, straight flush 4000, and royal flush is 4500.

The number of cards dealt is 5 cards since the type of game we are playing is 5 cards draw poker. Five cards draw is the standard poker game. The five cards are dealt to each player face down. Each player can exchange five cards on the draw, and each player is allowed to have no more than three times draw. In this project, player has to click 'DEAL' to draw cards. Normally, the cards are dealt (usually five cards), followed by a betting round, followed by the draw (or deal in this project), followed by a betting round, followed by a draw (or deal in this project), and followed by a final betting round.

**Ranking of the hands:**

1) High Cards. All five cards are just a bunch of individual cards. If no one has anything better such as one pair, two pair, three of a kind, full house, straight, flush, four of a kind, and straight flush etc. as we describe below, a single high card can win a hand, or if two people have the same pair, the player holding the highest other card wins. If two people have the same highest high card, you look

at the next highest card in their hands. For example, A-Q-2-3-4 beats A-J-10-9-8, and 10-9-8-6-3 beats 10-9-8-6-2.

2) One pair. A hand that contains a pair beats a hand that does not contain one. So a hand full of small cards like 2-2-3-4-5 beats a hand like A-K-Q-J-9, because that A-K-Q-J-9 hand is just a bunch of individual cards. The pair of twos wins. If both players have one pair, the player with the higher pair wins, regardless of the other three cards; for example, 7-7-2-3-4 beats 4-4-A-K-Q. If both players have the same pair, the next highest card decides the hand, for example, J-J-8-4-2 beats J-J-7-6-5. Similarly, J-J-8-4-3 would beat J-J-8-4-2.

3) Two Pairs. Any hand containing two pairs beats any hand containing one pair. For example, 2-2-3-3-6 beats A-A-K-Q-J. If both players have two pair, the hand containing the higher pair wins, for example, K-K-8-8-6 beats Q-Q-J-J-10. If players have the same top pair, the second pair decides it, for example K-K-8-8-6 beats K-K-7-7-Q, and if both players have the same two pair, the fifth card decides it, for example K-K-8-8-7 beats K-K-8-8-6.

4) Three of a Kind. Any hand containing three of a kind beats any hand containing two pairs, for example, 2-2-2-4-5 beats A-A-K-K-Q. If two players each have three of a kind, the higher three of a kind wins, for example 7-7-7-6-2 beats 5-5-5-A-K. We never see a situation where two people have the same three of a kind (For example: 4-4-4-A-K opposing 4-4-4-3-2, because there are only 4 Fours in a deck).

5) Straight. Five cards are in sequence. It defeats three of a kind, or any lower-ranked hand. For example, 2-3-4-5-6 beats A-A-A-K-Q. In the event that two

people hold a straight, higher straights beat lower straight, for example, 9-10-J-Q-K beats 3-4-5-6-7. Note that the Ace can play at either end: A-2-3-4-5 is a straight (the lowest possible straight), and also is 10-J-Q-K-A (the highest possible straight). But the Ace can't play in the "middle" of a straight. For example: a hand like Q-K-A-2-3 is not a straight. It is just a five-card hand.

6) Flush. Five cards are all in the same suit, for example, the 2-3-5-8-10 of hearts, defeat any straight or lower hand. If two people have flushes, the flush containing the highest card wins, for example, A-10-8-6-5 of hearts beats K-Q-J-9-7 of spades. If both flushes contain the same highest card, the second highest card decides it, for example, K-J-4-3-2 of diamonds beats K-10-9-8-4 of spades. If the first two cards are equal, the third card decides it, and so on. It is important that suits do not matter: if one player holds A-K-10-9-8 of diamonds and another holds A-K-10-9-8 of spades, they tie.

7) Full House. Five cards containing a three of a kind and a pair, for example, K-K-K-7-7 beats any flush or lower-ranked hand. If two people hold full houses, the full house containing the higher three of a kind wins, for example, Q-Q-Q-2-2 beats J-J-J-A-A. Just as we saw when examining three of a kind, we will not see situations like Q-Q-Q-5-5 opposing Q-Q-Q-8-8.

8) Four of a Kind. Five cards contain four of one card, for example, 4-4-4-4-6. If two people hold four of a kind, the higher "quads" wins. For example, 4-4-4-4-6 beats 3–3-3-3-A.

9) Straight flush. It is a very rare hand. Five cards are not only in sequence but also in the same suit, for example, 5-6-7-8-9 of diamonds, is a straight flush, and it

beats four of a kind or any lower-ranked hand. Just as in regular flushes, suits are irrelevant: 5-6-7-8-9 of diamonds splits the pot with 5-6-7-8-9 of spades. A Royal flush is just the best possible straight flush.

**Betting and Playing:**

The ante is the maximum money of the first bet. Another way to say that is a token amount of money that ensures that each player already has some stake in the game being played.

Players have three options in this project:

1. Calling: The real calling is to bet enough to match the amount bet since the last time the player placed a bet. For example, if the first player bets 10 cents and someone after the first player bets 15 cents, then the first player owes 5 cents to the pot. However, in this project, we check whether the player owes money to the pot after a player click CALLING button. If any player owes money to the pot, a message is displayed and tells the player to put more money in. It does evaluate all hands after all players have the same amount of money in the pot, and the best hand wins the money in the pot.

2. Raising: The amount of bet that a player wants to raise. In this project, user enters the amount of money that she or he wants to raise and clicks the "RAISING" button.

3. Folding: Drop out of the current hand. The player who does folding will not add money to the pot. In this project, we compare the rest of the players' hands, and the hand with the highest score wins.

When one player clicks either call, or fold, the best hand will win the money of the pot, and the rest of the players except the person who does fold lose the bet.

In implementation, an object of class card represents one of the 52 cards in a standard deck of playing cards. Each card has a suit and a value. An object of type Hand represents a hand of cards. The maximum number of cards in the hand is specified in the constructor, but by default it is 5. A utility function is provided for computing the value of the hand in the game of Blackjack. An object of type Deck represents an ordinary deck of 52 playing cards. The deck can be shuffled, and cards can be dealt from the deck.

Each player has a window to play the poker game. All information about the network poker game is stored in files in server side. For security season, we lock corresponding files by using lock( ) function when a player doing an action such as deal, calling, and folding etc, and a lock is released after the action is done by using free( ) function. In this way, we prevent multi user accessing a file at the same time.

```
void free( )
{
        File file = new File("busy.lock");
        System.out.println("Deleting busy.lock\n");
        file.delete();
        {
                File fileDest = new File("free.lock");
                try {
                        System.out.println("creating free.lock");
                        fileDest.createNewFile();
                        fw.close();
                }
                catch(Exception e){System.out.println(e);};
        }
}
void lock()
{
        File file = new File("free.lock");
        File fileDest = new File("busy.lock");
        while (!file.renameTo(fileDest))
        try
        {
                System.out.println("\nFile locked waiting....");
                Thread.currentThread().sleep(1000);
        }
        catch(Exception e){System.out.println(e);};
}
```

## 5. DEPLOYMENT

### 5.1 Test Data for Poker Hand

This test data is used to verify the correctness of the formula, which we used to distinguish different poker hand in section 4.7.

Data1: high card (king, Queen, Jack, 10, 8) and a pair (Ace, Ace, 2, 3, 4)

Score of high card = 154

Score of a pair = 517

Data 2: a pair (10, Jack, Queen, king, king) and two pairs (Ace, Ace, 2, 2, 3)

Score of a pair = 674

Score of two pairs = 1014

Data 3: two pairs (J, Q, Q, K, K) and three of a kind (A, A, A, 2, 3)

Score of two pairs = 1182

Score of three of a kind = 1510

Data 4: three of a kind (J, Q, K, K, K) and straight (Ace, 2, 3, 4, 5)

Score of three of a kind = 1686

Score of straight = 2030

Data 5: straight (10, J, Q, K, A) and flush (A, 2, 3, 5, 6 of hearts)

Score of straight = 2125

Score of flush = 2528

Data 6: flush (9, J, Q, K, A) and full house (A, A, A, 2, 2)

Score of flush = 2616

Score of full house = 3007

Data 7: full house (K, K, K, Q, Q) and four of a kind (A, A, A, A, 2)

Score of full house = 3186

Score of four of a kind = 3506

Data 8: four of a kind (K, K, K, K, Q) and straight flush (A, 2, 3, 4, 5 of diamonds)

Score of four of a kind = 3692

Score of straight flush = 4028

From the above scores we can see that the poker hands evaluating formula is correct. It gives us a correct ranking of poker hands.

## 5.2 User Experience

We let users test the network poker game to observe the result, which would help us improve the behavior of the simulated facial emotion in 3D graphic. We allowed the players to spend about 30 minutes testing the application after which we presented the players with a questionnaire. The questionnaire contained the following questions, which could be answered with a to f grade.

1) Ease of use
2) Quality of the application
3) Innovative grade

In addition we asked the users to write a single paragraph about their experience.

User experiences.

Average User grades: 1) c+ 2) b- 3) a-

User Opinion 1): The game is an interesting application of web interaction technologies. The framework looked solid. The graphical details less impressive, with some more work on the user interface needed.

User Opinion 2): This game is kind of fun, but the face images need to be improved. Such as more emotions and more like real human beings; therefore the players will get more interested in it. Anyway is a nice game, even though it’s a bit primitive.

## 6. ACKNOWLEDGEMENTS

I gratefully acknowledge the dedicated efforts of my master project’s advisor, Dr. Christopher Pollett at San Jose State University. This project was introduced and supported by him; the project would not have been finished without his guidance, patience and constant encouragement. I also thank Dr. Sin-Min Lee and Dr. Ho Kuen NG for having made available their time and commitment to serve on my committee.

## 7. CONCLUSION:

This project designed and implemented a network poker game with 3D-characters being able to express emotional information based on their cards. We developed an encoding scheme for recording human facial expressions by using DTD, which is XML-based language. Using XML application to produce realistic facial emotional character agents in 3D graphics, we developed an XML program for applying emotion availabilities for each player. The program allows users to play real poker game through the Internet and display facial emotion for a each player based on the player’s cards and emotion availabilities.

We successfully used the Document Object Model to parse player's XML file to get the emotion availabilities of each player. Java Servlet is used for web programming to interact with the user, control the browser, and dynamically create HTML content. Displaying 3D graphic in web programming is a challenging task. We have successfully used JavaScript to open VRML mime type and display 3D VRML of players' faces and poker cards. We used RequestDispatcher in servlet to allow a user to send emotion and cards' information to the server so that the server can display the corresponding 3D emotion and cards. Since all information is saved in files on the server side, it takes a longer time to get information back and forth, which is the weakness of the project, and it might be improved in a feature version.

## FUTURE WORK

As we described in section 4.1, there are many facial emotion availabilities in player's DTD, and we only implemented several facial emotion availabilities in 3D graphic. The other facial emotion availabilities are left for future work.

In section 3.8, we mentioned a dynamically remove operation in future versions which will remove the existing game when a user does not need it and will remove all data files after a game is over.

There was not enough time to implement a viewpoint for each player so that a user can see players' emotion easily and clearly. We also left this part as feature work.

## 8. REFERENCES:

1. [E41] Efron, D. Gesture and Environment. King's Crown Press. 1941.
2. [EF69] Ekman, P. and Friesen, W.V. The repertoire of nonverbal behavioural categories: Origins, usage, and coding. Semiotica 1:49-98.
3. [EF78] Ekman, P. & Friesen, W. V. The Facial Action Coding System (FACS): A technique for measurement of facial action. Consulting Psychologists Press. 1978.
4. [M92] David McNeill. Hand and Mind. University of Chicago Press. 1992.
   [W3C97] Extensible Markup Language (XML). www.w3.org/XML.
5. [W3C99] XSL Transformations (XSLT) Version 1.0. www.w3.org/TR/xslt.
6. [ZHMI00] Zong, Y., Hiroshi, D. and Mitsuru, I. Emotion expression functions Attached to Multimodal Presentation Markup Language MPML. 4th International Conference on Autonomous Agents. 2000.
7. [ECMA99] Standard ECMA-262 ECMAScript Language Specification 3rd ed. http://www.ecma.ch/ecma1/stand/ecma-262.htm. ECMA. 1999.
8. [W1] http://crl.research.compaq.com/projects/facial/facial.html
9. [WL94A] Waters, K. and Levergood, T. DECface: An automatic Lip-Synchronization Algorithm for Synthetic Faces, CRL Technical Report 93/4, September 1994
10. [WL94B] Waters, K. and Levergood, T. An Automatic Lip-Synchronization Algorithm for Synthetic Faces, ACM Multimedia, San Francisco, pages 149-156, October 1994
11. [WL95] Waters, K., Levergood, T. DECface: A System for Synthetic Face Applications, Journal of Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 1, No. 4, pages 349-366. Nov. 1995
12. [FK96] Computer Facial Animation, Fred I. Parke and Keith Waters. A. Peters Ltd., Boston Massachusetts, ISBN 1-56881-014-8 Hardcover, pp. 450, 1996. [software available for download]
13. [WLM98] Waters, K, Rehg, J, Loughlin, M, Kang, S, Computer Vision for Human-Machine Interaction, Roberto, C, and Pentland, A (eds)., Cambridge University press, pages 83-96, 1998
14. [W92] Waters, K. Modeling Three-Dimensional Facial Expressions, Processing Images of Faces, Bruce and Burton eds., Ablex, New Jersey, pages 202-227, 1992
15. [AWL99] Dustin R. Callaway. Inside Servlets, Addison Wesley Longman, Inc,

    1999