Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Reporte final del proyecto Terminal

Transición de objetos compartidos entre espacios privado y público en un pizarrón compartido

Agustín Morales García 206205870

Asesor: María Lizbeth Gallardo López
Profesor-Investigador
Asociado D

Mayo 2011

Índice de contenido

Índice de Figuras

Índice de tablas

Motivación

Descripción del Problema

Un pizarrón colaborativo es una aplicación de dibujo en proceso de creación, donde será posible crear y editar dibujos a partir de figuras geométricas básicas y en donde varios usuarios podrán trabajar sobre la edición de estas figuras, a las cuales denominamos objetos; cada uno de los usuarios podrá estar conectado a la aplicación a través de una red de dispositivos a saber: PC, laptop, PDA o teléfono.

En este proyecto partimos de una aplicación de dibujo distribuida llamada pizarrón colaborativo realizada por Gabriela Sánchez Morales estudiante de Doctorado del CINVESTAV, esta aplicación cuenta con un espacio de trabajo para realizar dibujos a partir de una barra de herramientas. El objetivo de nuestro proyecto es dotar a la aplicación pizarrón colaborativo de 1) un espacio privado, en donde cada usuario realice sus objetos, independientemente del dispositivo que emplée, 2) un espacio público, donde cada usuario pueda compartir sus objetos con los otros usuarios. Para cualquier dispositivo, la interfaz muestra ambos espacios como zonas rectángulares identificadas por un color o una etiqueta (privado/publico). Todo objeto creado en el espacio privado pasará al espacio público a través de cualquiera de las siguientes acciones del usuario: 1) al arrastrar el objeto hacia el espacio público; 2) al hacer un doble click sobre el objeto; o 3) al seleccionar desde el botón derecho la opción de compartir.

Objetivo general

Diseñar e implementar un espacio de trabajo privado y un espacio de trabajo público (colaborativo) para una aplicación de dibujo llamada “pizarrón compartido”. Ambos espacios de trabajo podrán ser desplegados para cada uno de los miembros de un grupo, independientemente del dispositivo que empléen para trabajar con la aplicación de dibujo.

Objetivos particulares

- Diseñar un algoritmo que proporcione la propiedad de plasticidad para los espacios de trabajo público y privado de la aplicación “pizarrón compartido”.
- Implementar el algoritmo que proporcione la propiedad de plasticidad para los espacios de trabajo público y privado.
- Realizar pruebas sobre los espacios público y privado, con la propiedad de plasticidad, en un dispositivo PDA y en una PC.
- Elaborar manuales de usuario y manuales de instalación, de los espacios público y privado.
- Elaborar el reporte final.

Antecedentes

Las computadoras en nuestro hogar u oficina son ahora un medio de interacción con otras personas [1], por esta razón, aplicaciones del tipo *groupware*[1] son requeridas. Este tipo de aplicaciones soporta la cooperación de personas ubicadas en el mismo lugar o en lugares remotos. En este sentido, los PDA's juegan un papel importante, ya que son dispositivos personales con movilidad realmente efectiva.

Hoy en día, existen aplicaciones *groupware* que funcionan para PC's, pero son pocas las que funcionan en PDA's [2]. Un ejemplo de aplicación que funciona en PC es Grove, el cual permite realizar trabajo colaborativo sobre la edición de un documento en línea. Para llevar una aplicación colaborativa a cualquier dispositivo es necesario dotarlas de "plasticidad"; es decir, lograr que la aplicación se adapte adecuadamente 1) al tamaño de la pantalla del dispositivo, 2)la resolución de la misma pantalla y 3) a los recursos (memoria y procesador) del dispositivo, entre otros.

Este proyecto terminal forma parte de un trabajo de doctorado que está realizando la M.C. Gabriela Sánchez Morales en el CINVESTAV[2]. El trabajo de la M.C. Sánchez se enfoca en el Trabajo Cooperativo Asistido por Computadora (TCAC o CSCW en inglés). El CSCW permite a un grupo de colaboradores participar cooperativamente en la producción de entidades compartidas e intercambiar mensajes para coordinarse, aún cuando ellos no puedan reunirse físicamente en el mismo lugar y/o al mismo tiempo [3].

Los sistemas de software son cada vez más grandes; por lo tanto, ya no es posible trabajar secuencialmente: definir primero el problema completo, luego diseñar la solución, construir el software y finalmente, probar el producto. Es necesario un enfoque iterativo, que permita una comprensión creciente del problema a través de refinamientos sucesivos, llegando a una solución efectiva luego de múltiples iteraciones acotadas en complejidad [4]. RUP[3] utiliza y soporta este enfoque iterativo que ayuda a minimizar los riesgos de tipo: tiempo de desarrollo, cumplimiento de objectivos, etc., mediante la producción de prototipos ejecutables progresivos y frecuentes que promueven la retroalimentación por parte del usuario.

Justificación

Un pizarrón compartido es una aplicación de dibujo (parecida al paint brush de Microsoft) donde varios usuarios podrán trabajar sobre la edición de un dibujo (al cual denominamos objeto). Cada uno de los usuarios podrá estar conectado a la aplicación a través de distintos dispositivos, a saber: PC, laptop, PDA o teléfono. La finalidad de este proyecto fue dotar a la aplicación de dibujo con tres

1 Sistemas basados en computadoras que soportan gente comprometida en una meta en común y proporcionan un ambiente compartido.

2 Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

3 La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

Inicio, El Objetivo en esta etapa es determinar la visión del proyecto.

Elaboración,En esta etapa el objetivo es determinar la arquitectura óptima.

Construcción, En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.

Transmisión, El objetivo es llegar a obtener el release del proyecto.

funcionalidades: 1) un espacio privado, en donde cada usuario realice sus objetos, independientemente del dispositivo que empleé; 2) un espacio público, donde cada usuario pueda compartir sus objetos con los otros usuarios; y 3) Ambos espacios deben tener la propiedad de platicidad, es decir, permitir ser empleadas desde diversos dispositivos; por ejemplo, computadoras personales, laptos, PDA's y/o smarth phones

Este proyecto terminal fue pensado para una aplicación de dibujo sencilla; sin embargo, servirá de base para realizar una aplicación más elaborada, como lo sería un editor de textos colaborativo. Actualmente, el trabajo colaborativo demanda disponer de dispositivos móviles y aplicaciones que funcionen adecuadamente sobre ellos, a fín de que un usuario pueda colaborar en donde sea y a la hora que sea, en proyectos académicos o en proyectos empresariales.

Desarrollo del proyecto

Metodología de base empleada en el proyecto

Este proyecto se realizó utilizando un modelo iterativo e incremental. (ver Figura 1) Generalmente, se emplea cuando los requisitos son medianamente bien conocidos pero no son completamente estáticos o definidos. En la etapa de análisis, debe distinguirse los módulos del sistema, ésto facilita su desarrollo por incrementos. Pueden no existir una prioridad en los módulos por parte del cliente, pero el desarrollador debe fijarlas de todos modos con algún criterio. Con cada incremento se agrega un nuevo módulo, cubriendo los requisitos del sistema, o bien mejorando la versión previamente implementada. La siguiente figura muestra en un diagrama de bloques sobre el modelo y el flujo que se puede seguir para emplearla en un problema específico.

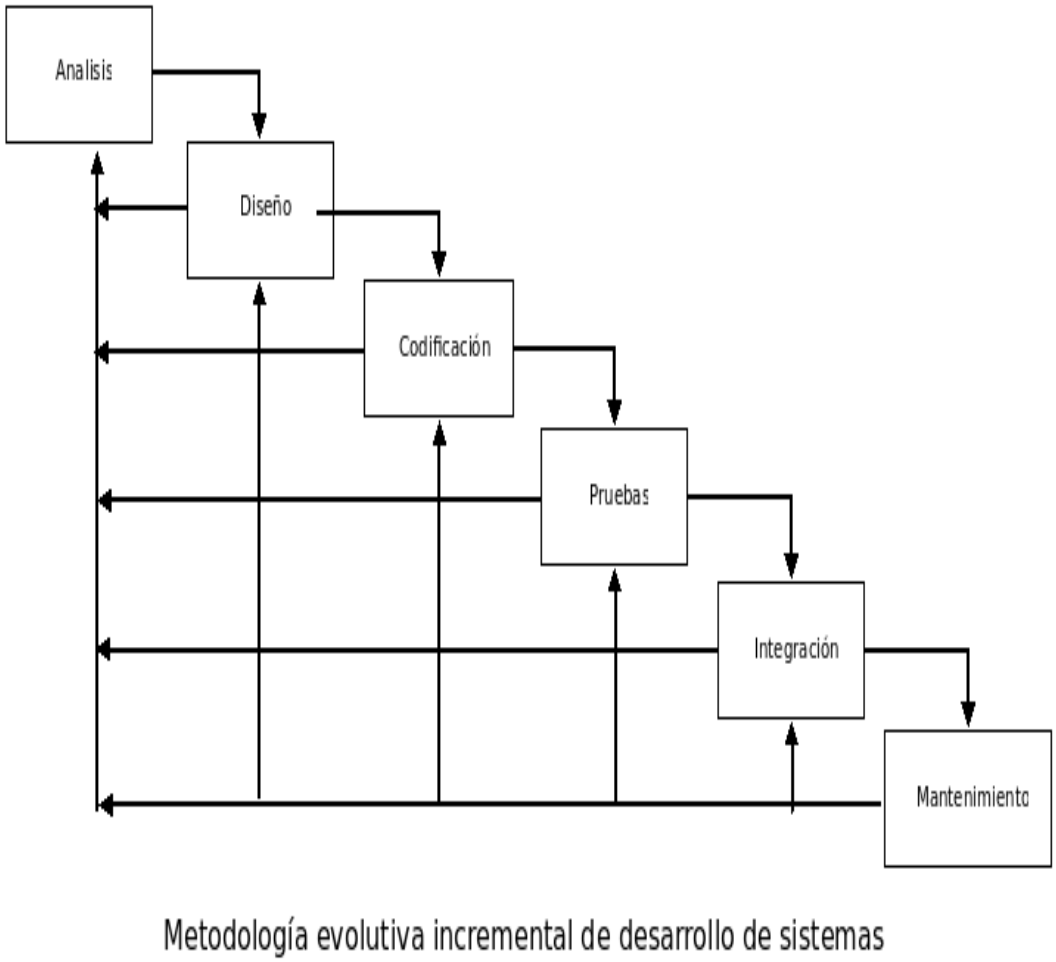


Figura 1: Modelo evolutivo

Diseño

En esta etapa del ciclo de desarrollo del sistema se crearon: la arquitectura del software, el diagrama de casos de uso, los escenarios de uso, el diagrama de clases y los mecanismos de plasticidad para la aplicación. Los diagrama se describen con mayor detalle en el documento de diseño de la aplicación. Durante el diseño realizamos varias iteraciones tratando de que el sistema sea lo más intuitivo posible para el usuario final y para el administrador.

Arquitectura del software

A continuación presentamos la forma de como esta construida la aplicación. Como se observa en la Figura 2, el sistema esta constituido por bloques que abstraen la presentación del sistema, el modelo del dominio de la aplicación, el control de las entidades que forman parte del modelo, así como el mecanismo de persistencia empleado. El bloque de presentación (GUI) es muy importante, dado que el sistema tiene que ver con una aplicación de dibujo; los elementos de esta interfaz son: 1) los espacios

de dibujo, y 2) los menús desplegables para un mejor control de la aplicación por parte del usuario. El bloque modelo contiene la lógica del negocio, es decir todas las clases que pueden abstraer las figuras para la aplicación. El bloque de control contiene toda la funcionalidad de la aplicación, es decir, todas las clases que nos permiten dibujar, re-dibujar, calcular las dimensiones o calcular los tamaños de las figuras. Y por último vemos un archivo xml el cual es el encargado de guardar los cambios que se realicen en la aplicación.

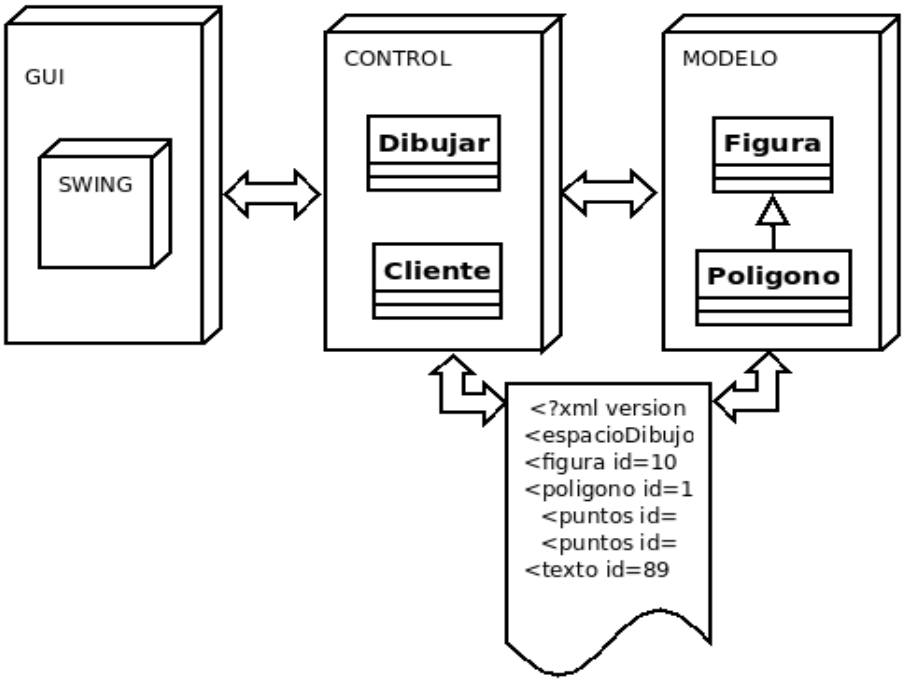


Figura 2: Arquitectura del software

Casos de uso y diagramas de diseño

La figura 2 muestra el diagrama general de casos de uso del sistema “pizarron colaborativo, espacio público/privado”; y luego, se presenta la descripción de cada uno de ellos.

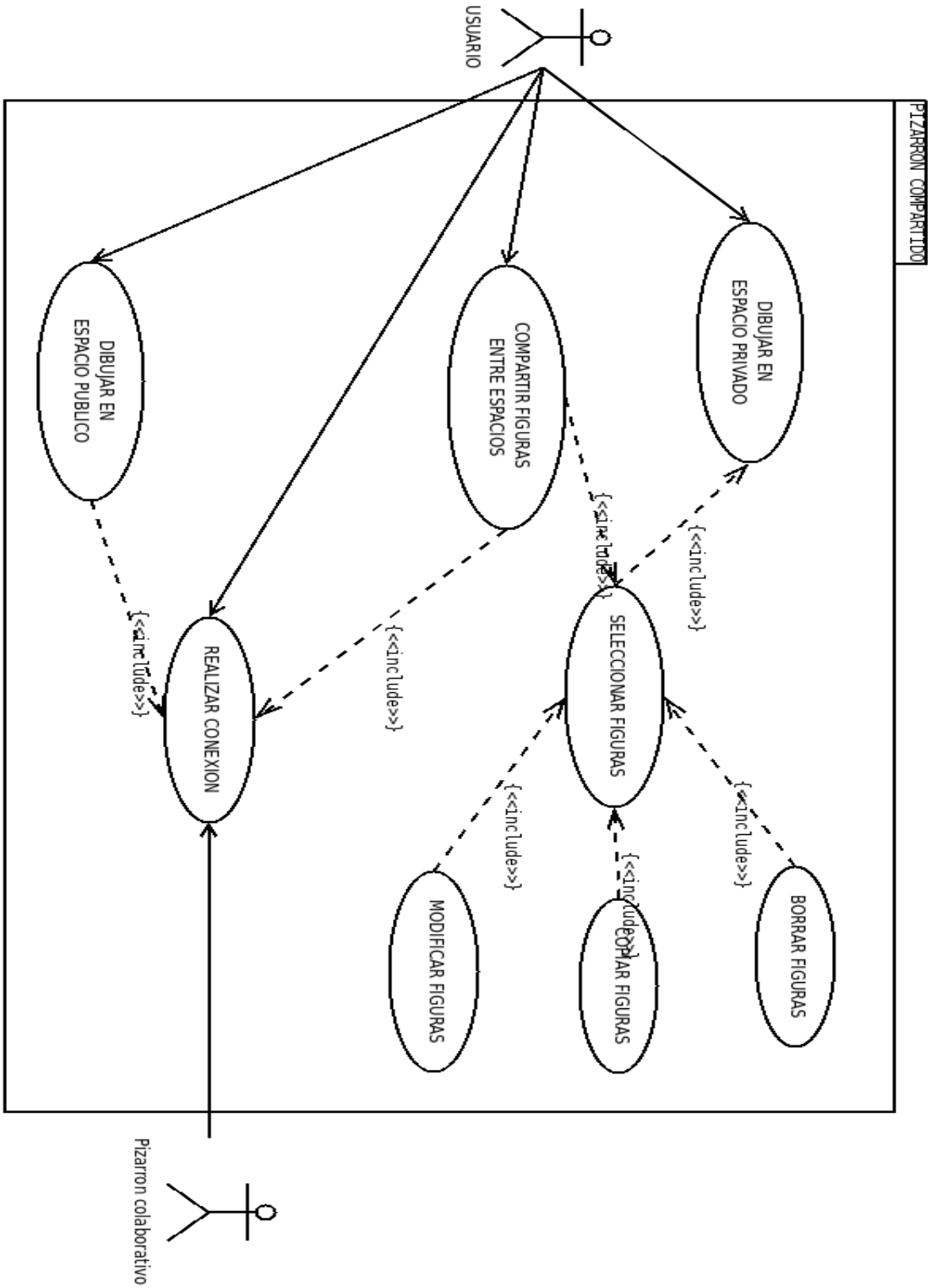


Figura 3: Diagrama de casos de uso

A continuación se describen cada uno de los casos de uso del sistema.

Caso de Uso: Dibujar
El usuario arrastrará el ratón por el espacio de dibujo para seleccionar el tamaño y origen de la figura, este caso de uso involucra varios eventos y varias funciones para determinar que figura se quiere dibujar, y en que área física se desea hacerlo (espacio público o espacio privado).

Tabla 1: Caso de uso dibujar

Caso de Uso: Borrar Figuras
El usuario puede borrar uno o varios objetos, siempre y cuando exista un objeto (figura, polígono, texto). Para que este caso de uso se lleve a cabo, deben estar visibles las áreas de dibujo; además, debe existir al menos un objeto dibujado en cualquiera de las áreas. Si se desean borrar varios objetos al mismo tiempo se debe seleccionar la opción de seleccionar.

Tabla 2: Caso de uso Borrar Figuras

Caso de Uso: Copiar figuras
El usuario puede copiar uno o varios objetos, siempre y cuando exista un objeto (figura, polígono, texto). Para que este caso de uso se lleve a cabo, deben estar visibles las áreas de dibujo y debe existir al menos un objeto dibujado en cualquiera de las áreas. Si se desean copiar varios objetos al mismo tiempo se debe seleccionar la opción de seleccionar.

Tabla 3: Caso de uso Copiar figuras

Caso de Uso: Modificar figuras
El usuario puede modificar un objeto cambiando su color, tamaño de borde, color de borde.

Tabla 4: Caso de uso Modificar figuras

Caso de Uso: Realizar Conexión
En este caso de uso intervienen el usuario y la aplicación pizarrón compartido. El usuario interviene al momento de ingresar la IP de la máquina donde esté corriendo un demonio (desarrollado específicamente) para realizar la comunicación. La aplicación pizarrón compartido realiza la conexión a un servidor, donde previamente debe estar en ejecución dicho demonio.

Tabla 5: Caso de uso Realizar Conexión

Caso de Uso: Dibujar en espacio privado
Este caso de uso contiene funciones que nos permiten administrar los eventos del ratón que permiten dibujar correctamente un objeto en el espacio privado.

Tabla 6: Caso de uso Dibujar en espacio privado

Caso de Uso: Dibujar en espacio público
Este caso de uso contiene funciones que permiten administrar los eventos del ratón para dibujar correctamente un objeto en el espacio público; además, permite administrar la conexión con otro equipo para compartir los objetos dibujados en este espacio de trabajo.

Tabla 7: Caso de uso Dibujar en espacio público

Caso de Uso: Seleccionar Figuras
Este caso de uso nos permite seleccionar varios objetos disponibles en la aplicación; estos objetos pueden ser seleccionados indistintamente del tipo que sean: figuras y texto.

Tabla 8: Caso de uso Seleccionar Figuras

Caso de Uso: Compartir Figuras
Este caso de uso permite compartir objetos con el espacio de trabajo público. Para que este caso de uso se lleve acabo debe haberse ejecutado correctamente el caso Realizar conexión. Este caso de uso se ejecuta cuando en el espacio público se ejecutan cualquiera de las siguientes funcionalidades: 1) se crea un nuevo objeto, 2) se modifica un nuevo objeto, 3) se elimina un objeto y 4) se comparte un objeto del espacio privado al espacio público y viceversa.

Tabla 9: Caso de uso Compartir Figuras

El diagrama de clases para el sistema “pizarrón colaborativo espacio público-privado”, derivado del análisis de los escenarios de uso, se muestra en la Figura 4. A continuación se describen brevemente cada una de las clases, las cuales están ordenadas de forma alfabética.

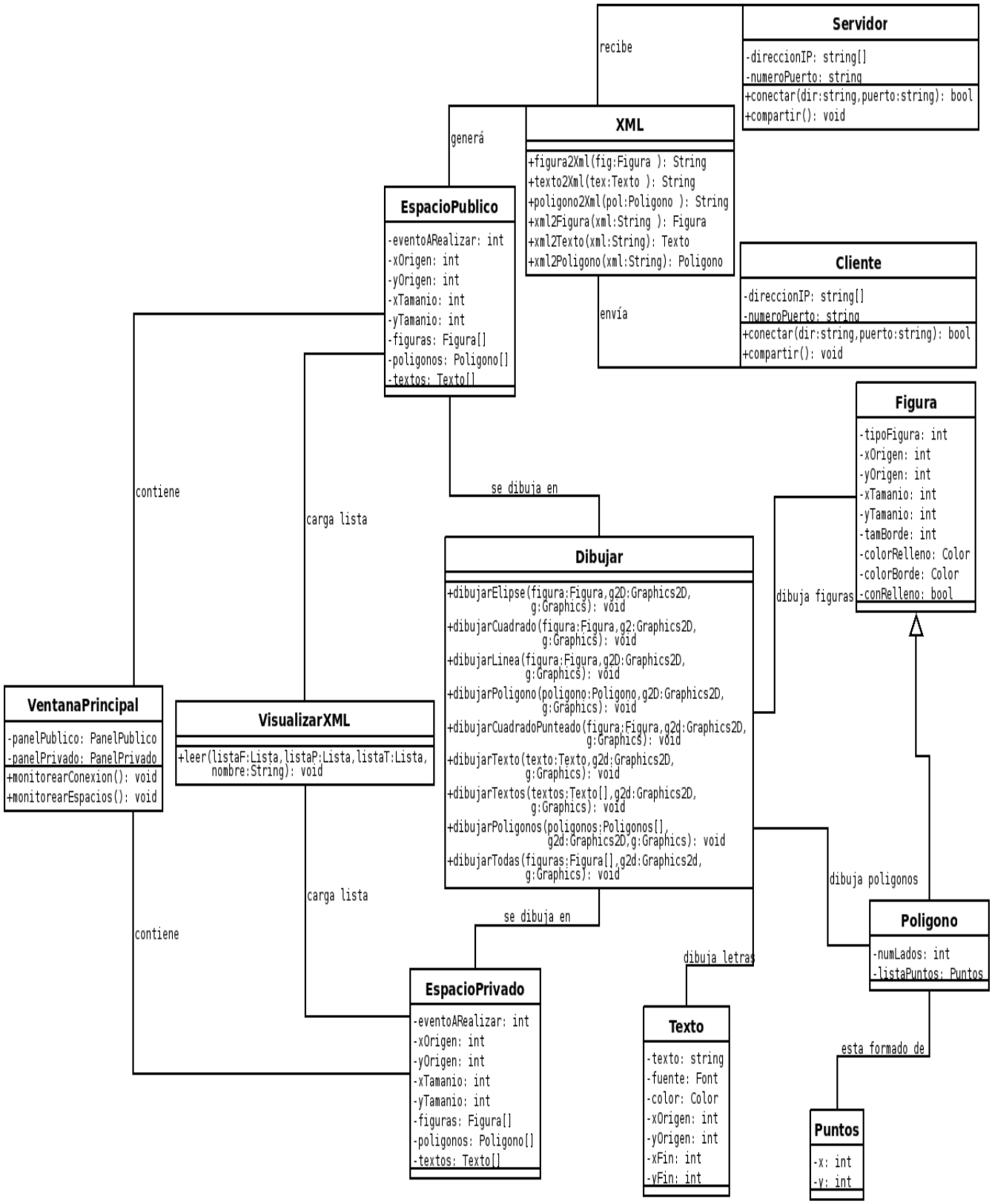


Figura 4: Diagrama de clases

- AgregarXML: Esta clase contiene funciones para agregar, eliminar y modificar objetos tipo Poligono, Texto y Figura, todas estas funciones actualizan el archivo XML donde se guarda las etiquetas que representan a los objetos que se pueden dibujar en cualquier espacio de trabajo. Este archivo tiene como objetivo mostrar, en una nueva sesión, los cambios efectuados en una sesión previa. Este archivo reside en el dispositivo cliente.

- Cliente: Esta clase interviene en la conexión que se crea para compartir objetos que se encuentran en el espacio público. Es la parte que actúa como cliente en el paradigma cliente-servidor. Envía una cadena de texto al Servidor, esta cadena contiene todos los atributos pertenecientes a los objetos que se pueden dibujar en cualquier espacio además esta cadena permite compartir cualquier objeto disponible en cualquiera de los espacios de trabajo.

- Servidor: Esta clase interviene en la conexión que se crea para compartir objetos que se encuentran en el espacio público de uno o varios usuarios. Sin embargo, esta clase no forma parte de la aplicación “pizarrón compartido”. Forma parte de otra aplicación llamada servidor que se programó para llevar a cabo la comunicación entre los diferentes dispositivos.

- Dibujar: Esta clase contiene todos los métodos para dibujar figuras (cuadro, círculo, línea, texto, polígono) sobre los espacios de trabajo público y privado. Esta clase interactúa directamente con la clase *graphics* de JAVA.

- EntradaDatos: Esta clase permite al sistema interactuar con el usuario para que éste ingrese, desde un *comboBox*, un nuevo tamaño para los objetos dibujados.

- EntradaTexto: Esta clase permite que el usuario ingrese un texto y elija las opciones del formato del texto que se va a dibujar en cualquiera de los espacio de dibujo.

- EspacioPrivado: Esta clase representa un espacio de trabajo privado con propiedades similares al espacio de trabajo público, la diferencia se centra en que este espacio no es visible para otros colaboradores, y es independiente de la conexión que se realiza entre los dispositivos. Esta clase es una de las más extensas, ya que desde esta clase se instancían todas las clases descritas en este proyecto; además esta clase maneja todos los eventos disponibles para dibujar una figura, o un texto.

- EspacioPublico: Esta clase representa un espacio de trabajo público donde los colaboradores pueden compartir su trabajo (figuras o texto) con otros colaboradores conectados en ese momento.

- Figura: Esta clase permite dibujar cuadrados, círculos y líneas; no así para el texto y el polígono, donde necesitamos la clase Poligono y Texto para representar mejor estos objetos. Es conveniente resaltar que inicialmente se pensó que esta clase podría abstraer todos los objetos que necesitamos dibujar en los espacios de trabajo público y privado; sin embargo, al estudiar más acerca del API de programación de JAVA, los métodos de la clase G*raphics* trata de manera diferente a los polígonos, el texto y los cuadrados.

- Poligono: Esta clase hereda de la clase Figura, y representa un polígono dibujado en los espacios de trabajo público o privado, un polígono puede ser de tres o más lados del mismo tamaño. Esta clase necesita a la clase Puntos para guardar los vértices del polígono. Así la clase Puntos se convierte en un atributo de la clase Poligono.

- Puntos: Esta clase representa solamente una coordenada (X,Y) en el espacio de trabajo tanto público como privado.

- Texto: Esta clase representa una cadena de texto pintada en el área de trabajo público o privado con propiedades de tamaño, tipo de letra y color de la letra. Para el tipo solo manejamos negrita, cursiva y negrita más cursiva.

- VentanaPrincipal: Esta clase representa una ventana principal de la aplicación; es aquí donde se deben instanciar e inicializar correctamente las clases EspacioPublico y EspacioPrivado; y además, es en esta clase donde se deben instanciar la barra de herramientas y la barra de colaboradores.

- VisualizarXML: Esta clase carga en memoria las listas de los objetos que se guardan en el documento XML; esta clase es llamada al momento de instanciar las clases EspacioPrivado y EspacioPublico, su función principal es leer el archivo XML y convertir e interpretar su contenido para crear instancias del tipo Figura, Poligono y Texto.

- XML: Esta clase es la encargada de convertir un objeto de los tipos Figura, Poligono y Texto a su representación XML en una cadena de texto y viceversa. Esta clase es utilizada por las clases Servidor y Cliente al momento de compartir objetos a través de la red.

Mecanismos de plasticidad

Para este proyecto, la plasticidad implica: 1) el intercambio de objetos entre espacios de trabajo público y privado; y 2) que los espacios público y privado puedan ser adaptados a distintos dispositivos, a saber: PC y dispositivos móviles. Hasta este punto del documento, hemos presentado el diseño de los espacios de trabajo público y privado; así como las clases para las figuras que se dibujan y se comparten entre esos espacios de trabajo. Todos estos elementos son la base para nuestro algoritmo de plasticidad que permita el intercambio de objetos entre los espacios de trabajo público y privado; El intercambio de objetos se realizo tanto para PC, como para dispositivos móviles, estos mismos elementos deberán servir para que este mismo algoritmo realice la adaptación de la aplicación a los diferentes dispositivos. La Tabla 10 presenta el algoritmo de plasticidad; posteriormente presentamos tres arquitecturas de comunicación que se experimentaron para lograr la comunicación entre los distintos dispositivos; y finalmente presentamos el mecanismo de persistencia de los datos a intercambiar entre los diferentes dispositivos.

```
Crear un espacio de trabajo privado.
Crear un espacio de trabajo publico.
Haer visibles los espacios en una ventana que proporcione una barra de herramientas, para seleccionar las
funcionalidades que proveen los espacios de trabajo.
Si dibujar objeto entonces
        Obtener coordenadas de los eventos del ratón
        xOrigen = min(xInicio, xFin)
        yOrigen = min(yInicio, yFin)
        xTamaño = abs(xInicio-xFin)
        yTamaño = abs(yInicio-yFin)
        dibujarObjeto(propiedades objeto)
        agregar objeto ala lista de objetos dibujados
        repintar()
Si copiar entonces
        Si hay objetos a copiar entonces
        copiar propiedades del objeto en memoria
        Si pegar entonces
                Si se han copiado objetos entonces
                        Obtener coordenadas de destino
                        dibujarObjeto(propiedades del objeto)
                        agregar objeto ala lista de objetos dibujados
                        repintar()

        Si cortar entonces
                Si hay objetos a cortar entonces
                        buscar el objeto en la lista
                        eliminar el objeto encontrado
                        repintar()

        Si seleccionar entonces
                Si hay objetos en la selección entonces
                        crear lista de objetos seleccionados

        Si compartir entonces
                Si hay objetos seleccionados entonces
                        Si los objetos se encuentran en el espacio público entonces
                                dibujar los objetos en el espacio privado
                        Si los objetos se encuentran en el espacio privado entonces
                                dibujar los objetos en el espacio público
                        repintar()
```

Tabla 10: Algoritmo de plasticidad

Comunicación entre los dispositivos

La comunicación se lleva a cabo por medio de *sockets; s*e experimentaron 3 formas para realizar la comunicación entre los dispositivos PC y móviles. El objetivo era tener una conexión punto a punto (*peer to peer*), lo cual implica que la aplicación cuente con un cliente y un servidor integrado. Una conexión punto a punto mantiene una mayor independencia -entre los dispositivos al momento de crear la conexión- que una conexión centralizada. A continuación se describe brevemente tres pototipos que se experimentaron en la comunicación.

1. Comunicación cliente-servidor centralizado entre PC

Para hacer este tipo de conexión necesitamos de la clase Cliente y otra clase Servidor, la clase Servidor es parte de una aplicación externa a los espacios de dibujo, pero que se implementó para llevar a cabo la comunicación (ver Figura 5). La instancia del cliente es la entidad que tendrá la funcionalidad de enviar las cadenas xml al servidor, el cual a su vez envía esta cadena xml a todos los clientes conectados; finalmente, el cliente es el encargado de convertir la cadena xml recibida en un objeto de tipo figura, polígono o texto.

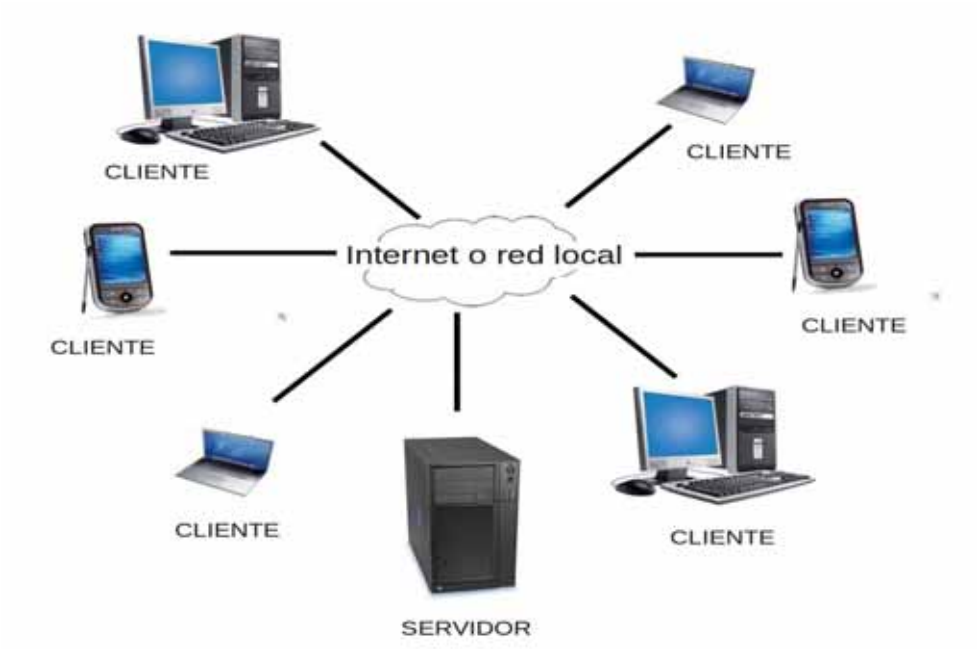


Figura 5: Interacción entre los dispositivos

El servidor funciona como un servidor de chat: 1) el servidor se encuentra en espera de una solicitud de un cliente; en cuanto el cliente se conecta, 2) se abre un nuevo canal de comunicación con el cliente; 3) el cliente envía una cadena de texto en xml; 3) el servidor reenvía la misma cadena a todos los clientes conectados en ese momento. El servidor funciona con dos hilos: uno encargado de recibir cadenas de un cliente y el otro encargado de reenviarla a todos los clientes conectados.

2. Comunicación distribuida (*peer to peer*) entre PC

Esta idea se llevó a cabo en la aplicación de escritorio y logramos realizar la conexión entre dos equipos PC conectados en una red local. La ventaja de esta solución es que no necesitamos una máquina dedicada para el servidor; y la desventaja es que necesitamos conocer las IP's de todos los equipos involucrados en la comunicación (ver Figura 6).

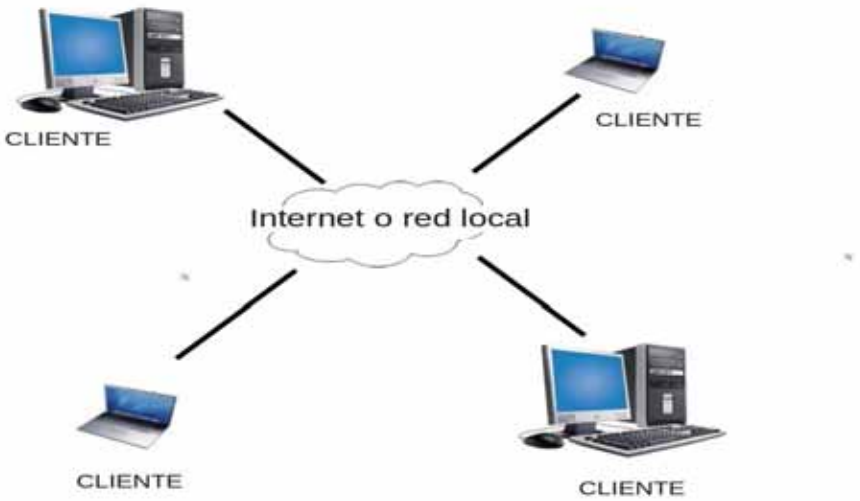


Figura 6: Comunicación distribuida

3. Comunicación distribuida (*peer to peer*) entre dispositivos móviles

Este tipo de comunicación se intentó implementar para dispositivos móviles (ver Figura 7), pero nos encontramos con varias limitantes tanto en el lenguaje JAVA como en la arquitectura del dispositivo; ya que J2ME no tiene las mismas clases que J2SE proporciona para implementar un servidor, y crearlas en J2ME implica más tiempo de lo que tenemos para desarrollar este proyecto. Por lo tanto el dispositivo móvil no puede ser cliente y servidor al mismo tiempo; en cuanto a las limitaciones físicas de los dispositivos nos encontramos limitada la capacidad del procesador 1000 MHz, por lo tanto el tiempo de cálculo es mayor comparado con una PC cuyo procesador es de 1600 MHz; la capacidad de almacenamiento primaria se reduce a 256Mb, cuando en una PC es de 2Gb; el tamaño de pantalla 600x480 pixeles, mientras que en una PC 1490x900 pixeles ausencia de botones y teclado en los dispositivos móviles sobre los cuales realizamos la experimentación.

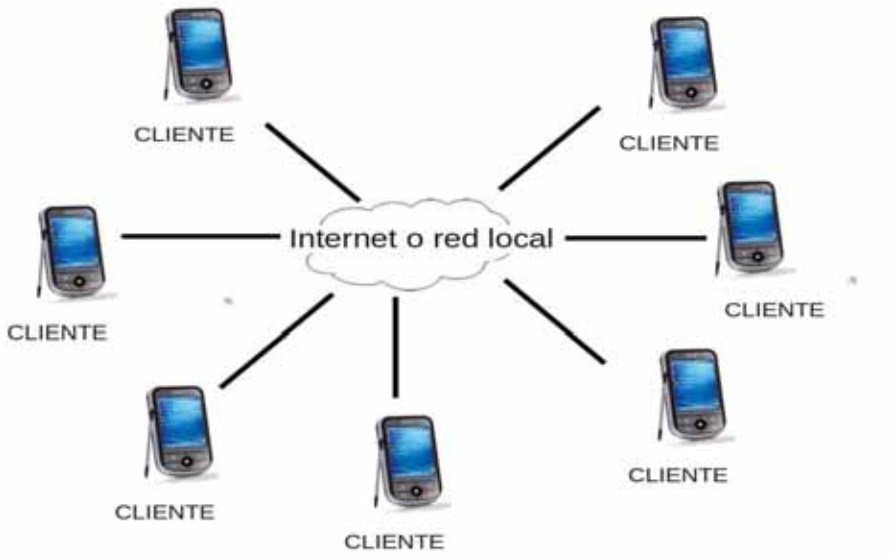


Figura 7: Comunicación distribuida entre móviles

Persistencia de los datos

XML, por sus siglas en inglés *eXtensible Markup Language* ('lenguaje de marcas extensible'), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C), que permite definir la gramática de lenguajes específicos. Algunos lenguajes que usan XML para su definición son: XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, también se ha propuesto como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo etc. XML hace posible que diversos sistemas compartan información de una manera segura, fiable y fácil. Por estas características se decidió utilizar XML como mecanismo para guardar en un archivo las características de las figuras dibujadas en los espacios público-privado. Al momento de realizar la comunicación entre un cliente y el servidor se envía una cadena XML como representación de un objeto dibujado en cualquier espacio de trabajo.

Para crear la estructura de los archivos que guardarán la información de una sesión de dibujo se emplearon un conjunto de etiquetas en XML específicas para polígono, figura, y texto. Donde la etiqueta figura puede guardar figuras como línea, cuadrado y elipse. La etiqueta polígono guarda todos los puntos que dan origen a un polígono. Todas estas etiquetas XML son una representación de las clases Poligono, Texto y Figura, donde sus atributos corresponden a etiquetas XML. En el manual técnico se detallan con más claridad estas etiquetas. A continuación mostramos una imágen con la estructura de las etiquetas usadas para definir un polígono en el archivo XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Document   : areaDeDibujo.xml
   Created on : 5 de febrero de 2011, 05:05 PM
   Author     : Agustin Morales
   Description:
      Purpose of the document follows.-->
<espacioDibujo>
        <figura id="0" idLista="1" tipoFigura="1" xOrigen="72" yOrigen="49" xTamanio="135"
        yTamanio="192" tamBorde="1" rojoRelleno="51" verdeRelleno="102" azulRelleno="255" rojoBorde="0"
verdeBorde="255" azulBorde="255" conRelleno="true" />

        <figura id="1" idLista="1" tipoFigura="2" xOrigen="22" yOrigen="289" xTamanio="112"
        yTamanio="421" tamBorde="1" rojoRelleno="204" verdeRelleno="255" azulRelleno="102" rojoBorde="0"
verdeBorde="255" azulBorde="255" conRelleno="true" />

        <poligono id="2" idLista="0" tipoFigura="5" xOrigen="357" yOrigen="371" xTamanio="395"
        yTamanio="411" tamBorde="1" rojoRelleno="255" verdeRelleno="0" azulRelleno="0" rojoBorde="0"
verdeBorde="255" azulBorde="255" conRelleno="true" xMinBorde="329" yMinBorde="323" xMaxBorde="412"
yMaxBorde="418" numLados="3">
                <punto id="0" x="412" y="371" />
                <punto id="1" x="329" y="418" />
                <punto id="2" x="329" y="323" />
                <punto id="3" x="411" y="371" />
        </poligono>

        <texto id="3" idLista="0" xOrigen="334" yOrigen="266" xFin="446" yFin="258" rojo="0" verde="0"
azul="0" fuente="Dialog.bold" tamFuente="8" nomFuente="Dialog.bold" t exto="hola que tal!!" />
</espacioDibujo>
```

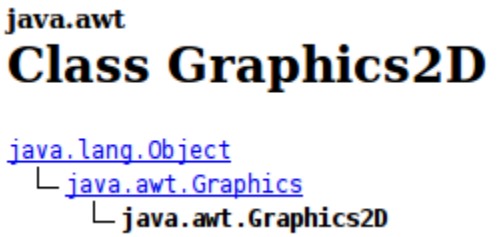
Tabla 11: Ejemplo de archivo XML

Implementación

El diseño que hemos presentado, a saber: escenarios de uso, modelo de clases, comunicación y la persistencia de los datos, fue implementado totalmente para PC; sin embargo, este mismo diseño se logró implementar parcialmente para dispositivos móviles, debido principalmente a las limitaciones que imponen estos dispositivos y al tiempo de desarrollo de este proyecto; concretamente para dispositivos móviles se implementaron los escenarios de uso y el modelo de clases.

Implementación de clases primitivas de JAVA

Un conjunto de clases que conforman el sistema “pizarrón compartido, espacio público-privado” heredan de clases primitivas de dibujo propuestas en JAVA; también creamos instancias de algunas de estas clases primitivas. La documentación de JAVA proporciona los métodos para pintar sobre un panel de dibujo. A continuación explicamos las clases primitivas que empleamos en el desarrollo de nuestro sistema.

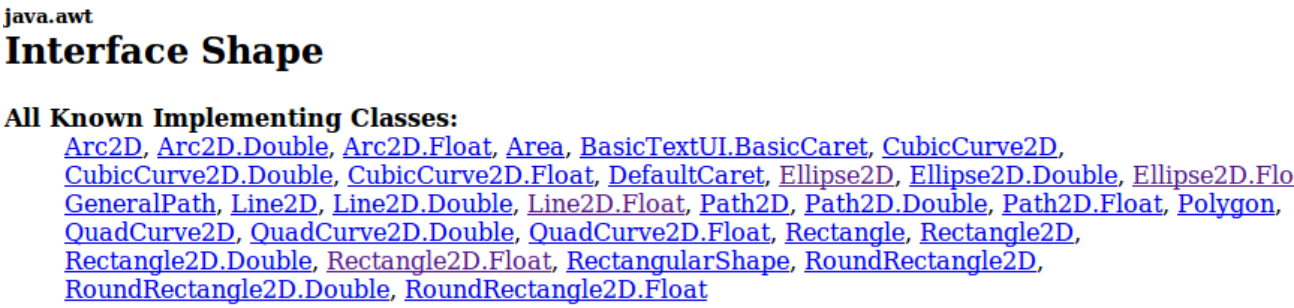


La clase Graphics2D hereda de la clase Graphics para proporcionar un mayor control sobre la geometría, transformaciones de coordenadas, gestión del color, y diseño de texto. Esta es la clase fundamental para las formas en dos dimensiones, texto e imágenes en Java. Los métodos que empleamos para el desarrollo de la aplicación son los siguientes:

`abstract void draw(Shape s)` la cual dibuja una figura con la configuración actual en el contexto `Graphics2D`.

`abstract void drawString(String str, int x, int y)` la cual dibuja la cadena de texto que se encuentra en str con la configuración actual en el contexto `Graphics2D`.

Para dibujar una figura necesitamos implementar la interfaz Shape la cual contiene a las clases que se utilizan para dibujar las figuras en un panel de dibujo.



De todas las clases posibles que tiene la interfaz `Shape`, utilizamos la clase `Ellipse2D`, `Ellipse2D.Float`, `Line2D, Line2D.Float, Rectangle2D, Rectangle2D.Float`, y la clase `Polygon;` estas clases se instanciaron llamando a su constructor con los parámetros creados con los distintos eventos que proporciona el *mouse* de la computadora. Por ejemplo, para dibujar un cuadrado una elipse o una línea necesitamos el ancho, el largo y la posición (X, Y) en donde se mostrará la figura, todos estos valores los obtendremos al manipular *mousePressed* y *mouseReleased*. El evento *mousePressed* toma la coordenada de donde se oprimió el botón izquierdo del ratón; el botón permanece oprimido, mientras se desplaza al ratón; y el evento *mouseReleased* toma la coordenada cuando se suelta el botón izquierdo del ratón.

Cuando se produce un evento del ratón JAVA nos proporciona una clase `MouseEvent` la cual contiene entre otras cosas dos métodos que nos sirven de manera adecuada para llevar a cabo nuestros objetivos; estos métodos son getX() y getY(), ambos regresan un número entero que corresponde a la coordenada (X ,Y) correspondiente al evento producido. Los valores que devuelven estas funciones los guardamos en variables llamadas xInicio, yInicio, xFin, yFin. Las variables xInicio y yInicio tendrán un valor cuando se produzca un evento del ratón del tipo *mousePressed;* y las variables xFin y yFin tomarán el valor devuelto por sus funciones getX y getY del evento tipo *mouseReleased*. Estos valores son los que se pasan al constructor de las figuras; pero para que la figura se pueda mostrar de manera adecuada en el panel de dibujo, hay que someterlas a operaciones de cálculo de distancias, funciones trigonométricas, entre otras.

Existen cuatro formas de obtener los valores de las coordenadas dependiendo del movimiento del cursor: 1) derecha-izquierda y de arriba-abajo, 2) izquierda-derecha y de arriba-abajo 3) izquierda-derecha y de abajo-arriba, 4) derecha-izquierda y de abajo-arriba. Como se muestra en la Figura 8.

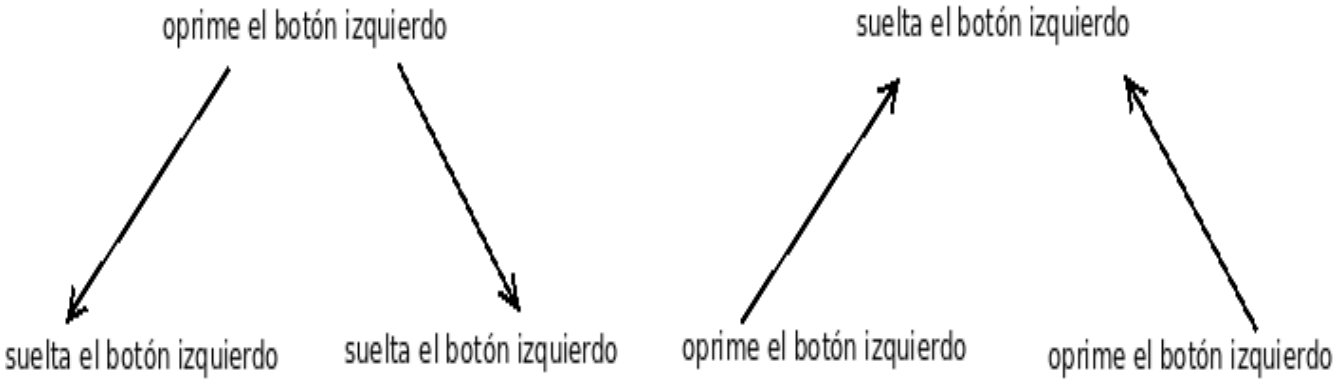


Figura 8: Distintos desplazamientos del ratón

Como se aprecia en la figura 9, el desplazamiento del ratón puede tener la misma magnitud, pero no así la misma dirección, es por eso que buscamos un patrón que se repita para así poder asignar los parámetros de forma adecuada. Vamos a suponer que se quiere dibujar una figura con las siguientes dimensiones 20 píxeles de ancho y 30 píxeles de altura, y que el punto donde comienza la figura sea en la coordenada X = 15 y Y = 27, esto se aprecia mejor en la figura 10.

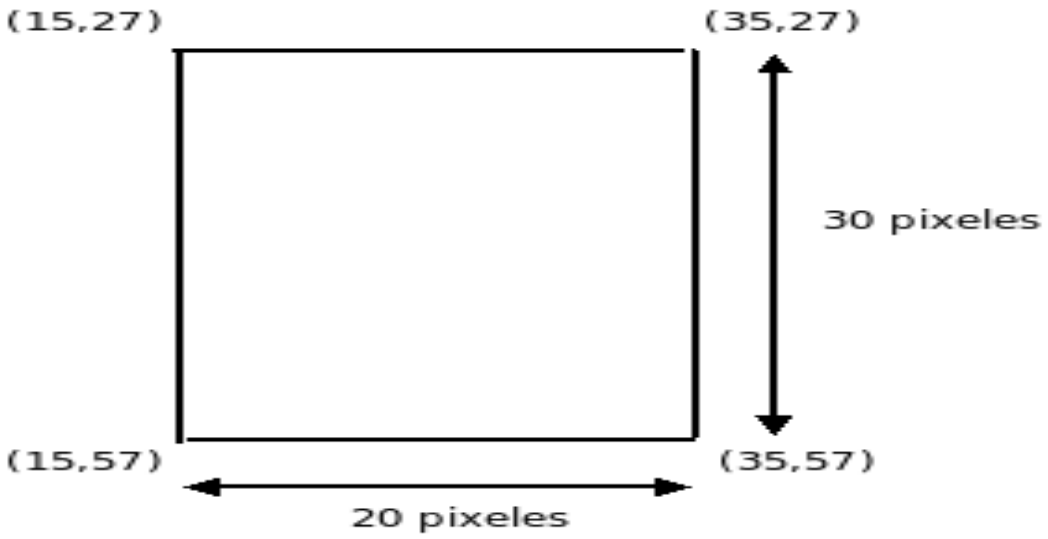


Figura 9: coordenadas de ejemplo para un dibujo de 20x30 píxeles

Comencemos por el primer desplazamiento que es de arriba-abajo y de derecha-izquierda: las variables tienen los valores xInicio=35, yInicio=27, xFin=15 y yFin=57. Pero nosotros queremos que la figura se comience a dibujar a partir de la coordenada (15,27) y estas coordenadas las podemos obtener de las variables, si buscamos el valor mínimo entre xInico y xFin y el valor mínimo entre yInicio y yFin, este valor mínimo será el origen de nuestra figura a las cuales nombraremos xOrigen y yOrigen, por lo tanto:

xOrigen = 15 = min(xInicio, xFin)
yOrigen = 27 = min(yInicio, yFin)

El tamaño en este caso tiene que ser de 20 pixeles de ancho y 30 píxeles de altura, si obtenemos la resta en valor absoluto de xInicio con yFin y la resta de yInicio con yFin tendremos lo siguiente:

xTamaño = 20 = abs(xInicio-xFin)
yTamaño = 30 = abs(yInicio-yFin)

Las fórmulas anteriores son las que nos ayudan a resolver el problema de los distintos tipos de desplazamiento por parte del usuario. Aquí se muestra de manera particular pero funciona para cualquier juego de coordenadas que resulten de los eventos del ratón. Para los siguientes tres casos restantes se aplican las mismas fórmulas, no importando de que esquina se oprima el botón del ratón y no importando en que esquina se suelte el botón del ratón.

Para el segundo caso: de arriba-abajo y de izquierda-derecha tenemos que xInicio = 15, yInicio = 27, xFin = 35, y yFin = 57, por lo que el resultado es:

xOrigen = 15 = min(xInicio, xFin)
yOrigen = 27 = min(yInicio, yFin)
xTamaño = 20 = abs(xInicio-xFin)
yTamaño = 30 = abs(yInicio-yFin)

Para el tercer caso: de abajo-arriba y de izquierda-derecha tenemos que xInicio = 15, yInicio = 57, xFin = 35, y yFin = 27, por lo que el resultado es:

xOrigen = 15 = min(xInicio, xFin)
yOrigen = 27 = min(yInicio, yFin)
xTamaño = 20 = abs(xInicio-xFin)
yTamaño = 30 = abs(yInicio-yFin)

Para el cuarto caso: de abajo-arriba y de derecha-izquierda tenemos que xInicio = 35, yInicio = 57, xFin = 15, y yFin = 27, por lo que el resultado es:

xOrigen = 15 = min(xInicio, xFin)
yOrigen = 27 = min(yInicio, yFin)
xTamaño = 20 = abs(xInicio-xFin)
yTamaño = 30 = abs(yInicio-yFin)

Con lo anterior se demuestra que la figura será del mismo tamaño y con el mismo origen. Sin embargo, estas fórmulas son válidas solo para dibujar figuras de tipo círculo, cuadrado y línea.

Para dibujar Texto solamente necesitamos los mínimos de cada coordenada en (X,Y).

Y para el polígono necesitamos hacer otro tipo de operaciones como las trigonométricas para poder dibujar, pero siempre nos apoyamos de las fórmulas anteriores para obtener xOrigen, yOrigen, xTamanio, yTamanio, ya que estos valores son las bases para dibujar cualquier figura, ya que provienen de los eventos que nos proporciona el puntero.

Ejemplos de algunos módulos programados para PC

Enseguida se muestran algunas funcionalidades programadas, aquellas que consideramos más importantes la función implementada que dibuja una elipse aplicando las fórmulas que se muestran arriba, para dibujar los cuadros y las líneas es similar, lo único que cambia es que no se creará un objeto Ellipse2D, si no que se creará un objeto Rectangle2D y un objeto Line2D respectivamente.

```
public static void dibujarElipse(Figura figura, Graphics2D grafico2D, Graphics g){
        Ellipse2D e2;
        Stroke bordeFigura;
        if (figura.isConRelleno()) {
           g.setColor(figura.getColorRelleno());
           g.fillOval(Math.min(figura.getxOrigen(), figura.getxTamanio()), Math.min
          ( figura.getyOrigen(), figura.getyTamanio()), Math.abs(figura.getxOrigen()  -
          figura.getxTamanio()), Math.abs(figura.getyOrigen() - figura.getyTamanio()));
        }
        e2 = new Ellipse2D.Float(Math.min(figura.getxOrigen(), figura.getxTamanio()),
          Math.min(figura.getyOrigen(), figura.getyTamanio()), Math.abs(figura.getxOrigen() -
          figura.getxTamanio()), Math.abs(figura.getyOrigen() - figura.getyTamanio()));
        bordeFigura = new BasicStroke(figura.getTamBorde(), BasicStroke.CAP_ROUND,
          BasicStroke.JOIN_MITER);
        grafico2D.setColor(figura.getColorBorde());
        grafico2D.setStroke(bordeFigura);
        grafico2D.draw(e2);

}
```

Tabla 12: Extracto de código de la función para dibujar un ovalo

A continuación se muestra la función implementada que dibuja un rectángulo con el objeto Rectangle2D.

```
  public static void dibujarCuadrado(Figura figura, Graphics2D grafico2D, Graphics g){

    Rectangle2D r2;
    Stroke bordeFigura;
    if(figura.isConRelleno()){
       g.setColor(figura.getColorRelleno());
       g.fillRect(Math.min(figura.getxOrigen(), figura.getxTamanio()),
             Math.min(figura.getyOrigen(), figura.getyTamanio()),
             Math.abs(figura.getxOrigen() - figura.getxTamanio()),
             Math.abs(figura.getyOrigen() - figura.getyTamanio() ));
    }
    r2 = new Rectangle2D.Float(Math.min(figura.getxOrigen(), figura.getxTamanio()),
          Math.min(figura.getyOrigen(), figura.getyTamanio()),
          Math.abs(figura.getxOrigen() - figura.getxTamanio()),
          Math.abs(figura.getyOrigen() - figura.getyTamanio() ));
    bordeFigura = new BasicStroke(figura.getTamBorde(),  BasicStroke.CAP_ROUND,     BasicStroke.JOIN_MITER);
    grafico2D.setColor(figura.getColorBorde());
    grafico2D.setStroke(bordeFigura);
    grafico2D.draw(r2);

  }
```

Tabla 13: Extracto de código de la clase Dibujar que dibuja un rectángulo

A continuación se muestra la función implementada que dibuja una linea con el objeto Line2D.

```
public static void dibujarLinea(Figura figura, Graphics2D grafico2D, Graphics g){

    Line2D r2;
    Stroke bordeFigura;
    if(figura.isConRelleno()){
       g.setColor(figura.getColorRelleno());
    }
    r2 = new Line2D.Float(figura.getxOrigen(), figura.getyOrigen(),
          figura.getxTamanio() , figura.getyTamanio() );
    bordeFigura = new BasicStroke(figura.getTamBorde(),  BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
    grafico2D.setColor(figura.getColorBorde());
    grafico2D.setStroke(bordeFigura);
    grafico2D.draw(r2);

  }
```

Tabla 14: Función de la clase Dibujar que nos permite dibujar una linea

Para dibujar un texto en el panel de dibujo simplemente se realiza llamando a la función *drawString*(String str, int x, int y), donde str es la cadena a visualizar; y los parámetros X,Y son las coordenadas, donde se va a posicionar el texto sobre el panel de dibujo.

La Figura 10 fue tomada del libro Como programar en JAVA [5], en la cual se muestran algunos elementos de las métricas de los tipos de letra. Los parámetros X,Y que se le envían a la función *drawString* corresponden a la esquina inferior izquierda de la línea base del tipo de letra.

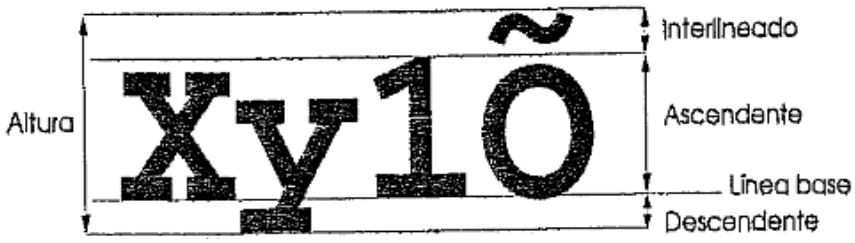


Figura 10: Métrica de tipos de letra

Por lo tanto, para dibujar un texto tomaremos en cuenta el evento del ratón *mouseClicked* que genera un *MouseEvent* (que incluye las funciones getX y getY) las cuales regresan la coordenada X,Y del punto donde se ha hecho clic con el botón izquierdo del ratón, este punto es el origen de nuestro texto que se envía a la función *drawString*.

```
public static void dibujarTexto(Texto texto, Graphics2D grafico2D, Graphics g){

    g.setColor(texto.getColor());
    g.setFont(texto.getFuente());
    g.drawString(texto.getTexto(),texto.getxOrigen(),texto.getyOrigen());

  }
```

Tabla 15: Función que dibuja un texto

El método que dibuja a un polígono es un poco más complejo, ya que tiene que calcular la distancia entre dos puntos, convertir de coordenadas rectangulares a polares; todo esto lo realiza en tiempo de ejecución, para pintar correctamente un polígono en cualquier área de dibujo. A continuación presentamos el método y describimos los cálculos efectuados.

```
public static void dibujarPoligono(Poligono poligono, Graphics2D grafico2D, Graphics g){
      Polygon s = new Polygon ();
      LinkedList<Puntos> puntos = new LinkedList<Puntos>();
           int x, y;
           float rad;
           int i;
           int r;
           int puntosLados=0;
      int xmax = Math.abs(poligono.getxOrigen() - poligono.getxTamanio());
      int ymax = Math.abs(poligono.getyOrigen() - poligono.getyTamanio());
      int xMinB = 1000000;
      int xMaxB = -1000000;
      int yMinB = 1000000;
      int yMaxB = -1000000;
           if(poligono.getNumLados()!=0){
        puntosLados = ( int ) ( 360 / poligono.getNumLados() );
        r = (int) Math.sqrt(xmax*xmax+ymax*ymax);
        for ( i = 0; i <= 360; i += puntosLados )
        {
           rad = ( float ) ( i * Math.PI / 180 );
           x = ( int ) ( poligono.getxOrigen() + Math.cos ( rad ) * r );
           y = ( int ) ( poligono.getyOrigen() + Math.sin ( rad ) * r );
           s.addPoint ( x, y );
           if(x<xMinB)
              xMinB = x;
           if(x>xMaxB)
              xMaxB = x;
           if(y<yMinB)
              yMinB = y;
          if(y>yMaxB)
              yMaxB = y;
           Puntos punto = new Puntos();
           punto.setX(x);
           punto.setY(y);
           puntos.add(new Puntos(punto));
        }
        Stroke bordeFigura;
        if(poligono.isConRelleno()){
           g.setColor(poligono.getColorRelleno());
           g.fillPolygon(s);
        }
        poligono.setPuntos(puntos);
        poligono.setxMaxBorde(xMaxB);
        poligono.setxMinBorde(xMinB);
        poligono.setyMaxBorde(yMaxB);
        poligono.setyMinBorde(yMinB);
 bordeFigura = new BasicStroke(poligono.getTamBorde(),  BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        grafico2D.setColor(poligono.getColorBorde());
          grafico2D.setStroke(bordeFigura);
        grafico2D.draw(s);
     }
  }
```

Tabla 16: Función que permite dibujar un polígono en un área de dibujo

Primero en la linea 16 obtenemos el número de puntos necesarios para dibujar el polígono, ésto se hace dividiendo 360 grados entre el número de lados para ver cuantas veces caben el número de lados en 360 grados, a este valor le llamamos puntosLados. Luego se obtiene el radio que debe de tener el círculo que circunscribe al polígono, este radio lo obtenemos a partir de la fórmula para obtener la distancia entre dos puntos. Dentro de un ciclo que va desde 1 hasta 360 con un incremento del valor de la variable puntosLados, vamos a hacer conversiones entre coordenadas polares y rectangulares, para obtener los puntos que permitirán dibujar el polígono.

Existen tres métodos más que permiten dibujar varias figuras, que reciben como parámetro una lista de figuras, una lista de polígonos o una lista de textos; estos métodos son útiles al llamar al método repaint() ya que reutilizamos el código empleado para dibujar un objeto por lo que no aumentamos el tamaño de la función paint(). Estas funciones lo único que hacen es que dentro de un ciclo iterativo(for) llaman a los métodos que dibujan un solo objeto en el área de dibujo como se aprecia en la Error: No se encuentra la fuente de referencia.

Esto se nos facilita ya que tenemos en memoria un LindedList del tipo Figura, un LinkedList del tipo Poligono y un LinkedList del tipo Texto. En estas listas ligadas se guardan todos los objetos que se dibujan en pantalla; además, de que en estas listas incluyen las operaciones eliminar o modificar cualquier objeto.

```
public static void dibujarTextos(LinkedList<Texto> listaTexto, Graphics2D grafico2D, Graphics grafico){
        for(int i=0; i<listaTexto.size(); i++){
          Dibujar.dibujarTexto(listaTexto.get(i), grafico2D, grafico);
        }
   }

 public static void dibujarPoligonos(LinkedList<Poligono> lista, Graphics2D grafico2D, Graphics g){
       for(int i=0; i<lista.size(); i++){
          Dibujar.dibujarPoligono(lista.get(i), grafico2D, g);
        }
   }

public static void dibujarTodas(LinkedList<Figura> listaFiguras, Graphics2D grafico2D, Graphics grafico){

       if(listaFiguras != null){
          for(int i=0; i<listaFiguras.size(); i++){
             if(listaFiguras.get(i).getTipoFigura() == EspacioPrivado.CIRCULO)
                Dibujar.dibujarElipse(listaFiguras.get(i), grafico2D, grafico);
             else if(listaFiguras.get(i).getTipoFigura() == EspacioPrivado.CUADRADO)
                Dibujar.dibujarCuadrado(listaFiguras.get(i), grafico2D, grafico);
             else if(listaFiguras.get(i).getTipoFigura() == EspacioPrivado.LINEA)
                Dibujar.dibujarLinea(listaFiguras.get(i), grafico2D, grafico);
          }
       }
   }
```

Tabla 17: Funciones que dibujan varios objetos provenientes de una lista

Todos los eventos del ratón, las funciones que dibujan las figuras, los polígonos y las cadenas de texto

se mandan llamar dentro de las clases EspacioPublico y EspacioPrivado, estas clase extienden de la clase JPanel la cual contiene todas las funciones necesarias para manipular los eventos del ratón.

Las clases EspacioPublico y EspacioPrivado son las que interactúan entre sí al momento de compartir los objetos entre espacios de trabajo. Esta interacción se da tomando en cuenta que en JAVA se manejan las referencias a los objetos, por lo tanto se creó una referencia del objeto EspacioPublico en el EspacioPrivado y una referencia del objeto EspacioPrivado en el EspacioPublico. Así cuando se quiera copiar un objeto del espacio privado al público, solamente se tendrá que manipular la referencia del espacio público agregando a su lista de figuras los objetos a compartir.

A continuación se muestra un fragmento de codigo del servidor que es el encargado de hacer el broadcast de la cadena de texto.

```
Socket scli=null;
    Socket scli2=null;
    DataInputStream entrada=null;
    DataOutputStream salida=null;
    DataOutputStream salida2=null;
    public static Vector<threadServidor> clientesActivos=new Vector();
    String nameUser;
    Servidor serv;
    public threadServidor(Socket scliente,Socket scliente2,Servidor serv)
    {
      scli=scliente;
      scli2=scliente2;
      this.serv=serv;
      nameUser="";
      clientesActivos.add(this);
      SimpleDateFormat fecha = new SimpleDateFormat("dd/MM/yyyy_HHmmss");
      Date date = new Date();
      serv.mostrar("Dibujante agregado: Dibujante"+fecha.format(date));
    }
    public void run()
    {
           serv.mostrar(".::Esperando Objeto :");
           try
           {
         entrada=new DataInputStream(scli.getInputStream());
       salida=new DataOutputStream(scli.getOutputStream());
       salida2=new DataOutputStream(scli2.getOutputStream());
       this.setNameUser("cliente # "+entrada.readUTF());
       enviaUserActivos();
           }
           catch (IOException e) {  e.printStackTrace();     }

      int opcion=0,numUsers=0;
      String amigo="",mencli="";
           while(true)
           {
       try
       {
         opcion=entrada.readInt();
         switch(opcion)
         {
           case 1://envio de mensage a todos
             mencli=entrada.readUTF();
             serv.mostrar("Se dibujara el sig Objeto: "+mencli);
             enviaMsg(mencli);
             break;
           case 2://envio de lista de uasuarios activos
             numUsers=clientesActivos.size();
             salida.writeInt(numUsers);
             for(int i=0;i<numUsers;i++)
               salida.writeUTF(clientesActivos.get(i).nameUser);
             break;
           case 3: // envia mensage a uno solo
             amigo=entrada.readUTF();//captura nombre de amigo
             mencli=entrada.readUTF();//mensage enviado
             enviaMsg(amigo,mencli);
             break;
         }
```

```
        }
        catch (IOException e) {System.out.println("El cliente termino la conexion");break;}
            }
            serv.mostrar("Se removio un usuario");
            clientesActivos.removeElement(this);
            try
            {
        serv.mostrar("Se desconecto un usuario");
        scli.close();
            }
      catch(Exception et)
      {serv.mostrar("no se puede cerrar el socket");}
    }

    public void enviaMsg(String mencli2)
    {
      threadServidor user=null;
      for(int i=0;i<clientesActivos.size();i++)
      {
        serv.mostrar("Objeto Devuelto:"+mencli2);
        try
         {

              user=clientesActivos.get(i);
           if(!user.getName().equals(this.getNameUser())){
              user.salida2.writeInt(1);//opcion de mensage
              user.salida2.writeUTF(mencli2);
           }
        }catch (IOException e) {e.printStackTrace();}
      }
    }
}
```

Tabla 18: Fragmento de código del servidor

Ejemplos de algunos módulos programados para J2ME

A continuación vamos a presentar algunas funciones que se programaron para los dispositivos que usan J2ME, así como las diferencias que hay entre la implementación para PC, aunque la lógica sea similar se implemento de manera distinta que para PC, esto debido a que algunas clases que se utilizaron en JAVA entandar no están disponibles para la versión J2ME.

La primera clase que se implemento fue la Clase Color ya que J2ME no maneja una clase que maneje los colores, pero como partimos de la implementación para PC y para no modificar demasiado el código creamos esta clase.

```
class Color {
   private int r;
   private int g;
   private int b;
   public Color(int r, int g, int b) {
      this.r = r;
      this.g = g;
      this.b = b;
   }   public Color() {
   }   public int getB() {
      return b;
   }   public void setB(int b) {
      this.b = b;
   }  public int getG() {
      return g;
   }   public void setG(int g) {
      this.g = g;
   }   public int getR() {
      return r;
   }   public void setR(int r) {
      this.r = r;
   }
}
```

Tabla 19: Clase Color para J2ME

A continuación en la Tabla 20 vamos a mostrar la función para dibujar una elipse y como puede apreciarse es mucho más pequeña que la función para dibujar en PC que se encuentra en la Tabla 12, ya que J2ME no tiene implementada la Clase Graphics2D que nos proporciona mayor funcionalidad como la funcionalidad de elegir el tamaño del borde de la figura, por lo que solo se utilizo la clase Graphics que si nos proporciona la API J2ME.

```
public static void dibujarElipse(Figura figura, Graphics g){
    if(figura.isConRelleno()){
        g.setColor(figura.getColorRelleno().getR(),figura.getColorRelleno().getG(),figura.getColorRelleno().getB());
                        g.fillArc(Math.min(figura.getxOrigen(), figura.getxTamanio()), Math.min(figura.getyOrigen(), figura.getyTamanio()), Math.abs(figura.getxOrigen() - figura.getxTamanio()), Math.abs(figura.getyOrigen() - figura.getyTamanio() ),0,360);
    }   g.setColor(figura.getColorBorde().getR(), figura.getColorBorde().getG(), figura.getColorBorde().getB());
    g.drawArc(Math.min(figura.getxOrigen(), figura.getxTamanio()),
        Math.min(figura.getyOrigen(), figura.getyTamanio()),  Math.abs(figura.getxOrigen() - figura.getxTamanio()),
        Math.abs(figura.getyOrigen() - figura.getyTamanio() ),0,360);
```

Tabla 20: Función para dibujar un elipse en J2ME

L o mismo sucede para dibujar la una línea y un cuadrado que se muestran en la Tabla 21 y la Tabla 22.

```
public static void dibujarLinea(Figura figura, Graphics g){
    if(figura.isConRelleno()){
      g.setColor(figura.getColorBorde().getR(), figura.getColorBorde().getG(), figura.getColorBorde().getB());
    }
    g.drawLine(figura.getxOrigen(), figura.getyOrigen(),
        figura.getxTamanio() , figura.getyTamanio() );
  }
```

Tabla 21: Función para dibujar una línea en J2ME

```
public static void dibujarCuadrado(Figura figura, Graphics g){
    if(figura.isConRelleno()){
       g.setColor(figura.getColorRelleno().getR(),figura.getColorRelleno().getG(),figura.getColorRelleno().getB());
       g.fillRect(Math.min(figura.getxOrigen(), figura.getxTamanio()),
            Math.min(figura.getyOrigen(), figura.getyTamanio()),
            Math.abs(figura.getxOrigen() - figura.getxTamanio()),
            Math.abs(figura.getyOrigen() - figura.getyTamanio() ));
    }
    g.setColor(figura.getColorBorde().getR(), figura.getColorBorde().getG(), figura.getColorBorde().getB());
    g.drawRect(Math.min(figura.getxOrigen(), figura.getxTamanio()),
        Math.min(figura.getyOrigen(), figura.getyTamanio()),
        Math.abs(figura.getxOrigen() - figura.getxTamanio()),
        Math.abs(figura.getyOrigen() - figura.getyTamanio() ));

  }
```

Tabla 22: Función para dibujar un cuadrado en J2ME

Para dibujar un polígono la principal diferencia que se muestra es que se utiliza la Clase Vector de J2ME para guardar los puntos de los vértices del polígono ya que J2ME no tiene implementada la función LinkedList que se utilizó para guardar los vértices del polígono en PC esto se aprecia en la Tabla 23.

```
  public static void dibujarPoligono(Poligono poligono, Graphics g){
   g.setColor(poligono.getColorBorde().getR(),poligono.getColorBorde().getG(),poligono.getColorBorde().getB());
   Vector puntos = new Vector();
         int x, y;
         float rad;
         int i;
         int r;
         int puntosLados=0;
   int xmax = Math.abs(poligono.getxOrigen() - poligono.getxTamanio());
    int ymax = Math.abs(poligono.getyOrigen() - poligono.getyTamanio());
    int xMinB = 1000000;
    int xMaxB = -1000000;
    int yMinB = 1000000;
    int yMaxB = -1000000;
         if(poligono.getNumLados()!=0){
       puntosLados = ( int ) ( 360 / poligono.getNumLados() );
       r = (int) Math.sqrt(xmax*xmax+ymax*ymax)/2;
       for ( i = 0; i <= 360; i += puntosLados )
       {
          rad = ( float ) ( i * Math.PI / 180 );
          x = ( int ) ( poligono.getxOrigen() + Math.cos ( rad ) * r );
          y = ( int ) ( poligono.getyOrigen() + Math.sin ( rad ) * r );
        //  s.addPoint ( x, y );
          if(x<xMinB)
             xMinB = x;
          if(x>xMaxB)
             xMaxB = x;
          if(y<yMinB)
             yMinB = y;
          if(y>yMaxB)
             yMaxB = y;
          Puntos punto = new Puntos();
          punto.setX(x);
          punto.setY(y);
          puntos.addElement(new Puntos(punto));
       }
    poligono.setPuntos(puntos);
       poligono.setxMaxBorde(xMaxB);
       poligono.setxMinBorde(xMinB);
       poligono.setyMaxBorde(yMaxB);
       poligono.setyMinBorde(yMinB);
       int j =0;
       for(j=puntos.size()-1;j>0;j--){
          g.drawLine(((Puntos)puntos.elementAt(j)).getX(), ((Puntos)puntos.elementAt(j)).getY(),
          ((Puntos)puntos.elementAt(j-1)).getX() , ((Puntos)puntos.elementAt(j-1)).getY() );
       }
     }
}
```

Tabla 23: Función para dibujar un polígono en J2ME

Ejecución de la aplicación

A continuación mostramos la ejecución del servidor y la del cliente la mostraremos en la siguiente sección mostraremos las pantallas y como se lleva el intercambio de cadenas y como es que estas cadenas se convierten en figuras pintadas en la aplicación.

En la Figura 11 se muestra la ejecución del servidor y podemos apreciar como es que esta esperando un nuevo cliente.

Figura 11: Ejecución del servidor

En la Figura 12 se muestra como se han agregado dos clientes a la conexión en la linea “Dibujante agregado” se muestra que el cliente con identificador Dibujante16/07/2011_202618 se agrego a la conexión y también se muestra que se agrego el cliente con identificador Dibujante16/07/2011_202948 y se muestra en la última línea que el servidor está disponible para un nuevo cliente, hay una línea más que nos dice “.:: Esperando Objeto:” que nos informa que el servidor que esta esperando que cualquier cliente realice alguna acción dentro de la aplicación

Figura 12: Servidor con dos clientes conectados

En la Figura 13 se muestra que un cliente compartió una figura en el espacio público y esta acción se señala con la línea “Se dibujará el sig Objeto”; luego, se imprime en pantalla la figura en su representación xml. Luego el servidor señala “Objeto Devuelto:”; esta línea se imprime en base al número de clientes que se encuentren conectados con el servidor, para esta demostración se encontraban dos clientes conectados y es por eso que se imprimió dos veces.

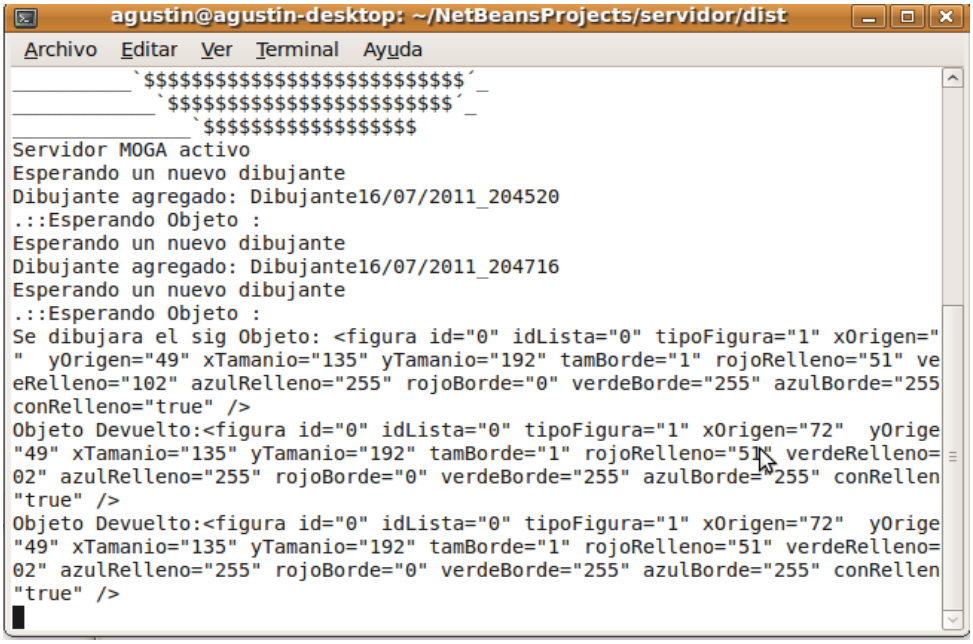


Figura 13: Dibujar Figura

Se envía una cadena xml junto con un comando; el cual indicará al cliente la acción que debe realizar. Los comandos de acción son los siguientes:

Comando	Acción que realiza.
<figura	dibujar una nueva figura
<poligono	dibujar una nuevo polígono
<texto	dibujar una nuevo texto
bordeFigura	modificar el borde de la figura indicada
bordePoligono	modificar el borde del polígono indicado
colorBordeFigura	modificar el color del borde
colorBordePoligono	modificar el color del borde
conFondoFigura	modificar la bandera de si es con fondo
conFondoPoligono	modificar la bandera de si es con fondo
sinFondoFigura	modificar la bandera de sin fondo
sinFondoPoligono	modificar la bandera de sin fondo
colorRellenoFigura	modificar el color de relleno
colorRellenoPoligono	modificar el color de relleno
arrastrarFigura	modificar la posición de la figura
arrastrarPoligono	modificar la posición del polígono
arrastrarTexto	modificar la posición del texto
cortarFigura	eliminar una figura
cortarPoligono	eliminar un polígono
cortarTexto	eliminar un texto
cortarFiguras	eliminar algunas figuras al mismo tiempo
cortarPoligonos	eliminar algunos polígonos al mismo tiempo
cortarTextos	eliminar algunos textos al mismo tiempo

Tabla 24: Palabras de acción en xml

Demostración de la aplicación funcionado

En esta sección vamos a mostrar la aplicación ejecutándose y los pasos que se deben realizar para que funcione correctamente; además mostraremos la parte de la aplicación que funciona en un dispositivo móvil pero ejecutándose en el emulador, esta ultima funcionalidad de la aplicación no se concluyó por completo ya que el dispositivo presenta muchas restricciones tanto en lenguaje como en recursos, los principales problemas que encontramos son: 1) J2ME no permite usar listas ligadas, 2) no cuenta con clases como las que encontramos en J2SE para manejar cadenas, ni para manejar xml. Por lo tanto, no se concluyó del todo la aplicación para dispositivos móviles, pero logramos dibujar figuras; falta por implementar la comunicación con el servidor.

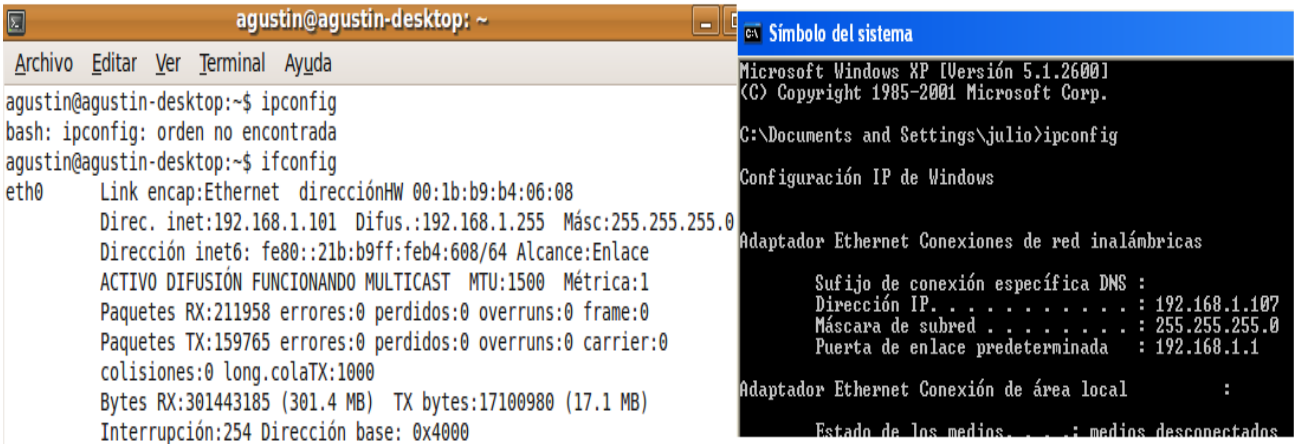


Figura 14: IP de dos clientes que se conectan al servidor

Para realizar la demostración primero vamos a mostrar dos imágenes donde se muestran las IP's de las máquinas que se conectaron al servidor. Vemos que la máquina que tiene el S.O. Ubuntu tiene la IP 192.168.1.101 y la máquina con Windows XP tiene la IP 192.168.1.107, ver Figura 14

Al correr la aplicación nos aparece una ventana donde nos pide ingresar la IP de la máquina donde se encuentra el servidor ejecutándose. Mostramos en la Figura 15 las ventanas que aparecen al ejecutar la aplicación en Ubuntu y en Windows respectivamente.

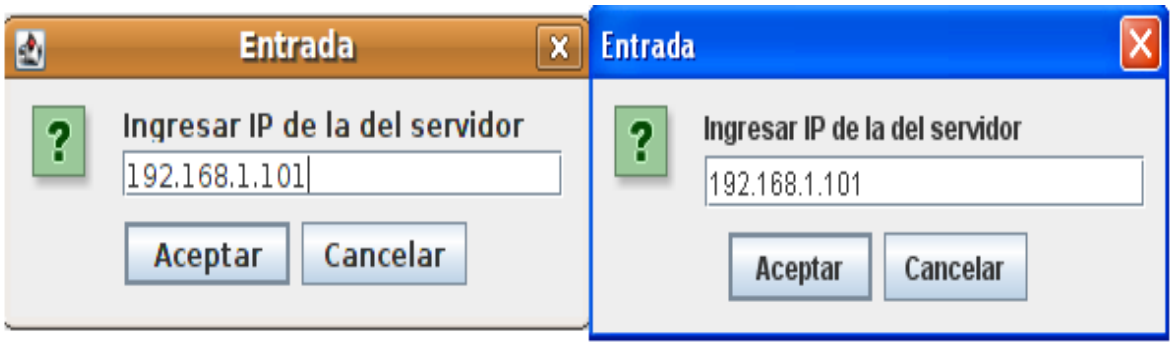


Figura 15: Entrada de la IP del servidor

Una vez ingresada la IP se muestran los dos espacios de trabajo el privado y el público. La aplicación carece de una barra de herramientas ya que no era el propósito de este proyecto, por lo que solamente se crearon botones que nos permitieran manejar los eventos durante las pruebas.

Vamos a mostrar en la siguiente figura que los dos clientes se han conectado y están preparados para compartir información: se cargan figuras que habíamos realizado en la sesión anterior, ésto se logra por la información de las figuras que se guardó en un archivo xml al cerrar la aplicación. En un principio cada que modificamos una figura guardábamos los cambios en el documento xml, pero la aplicación presentó algunos problemas de acceso de lectura por lo que decidimos que todos los cambios se realizaran al momento de cerrar la aplicación.

Aplicación ejecutada en Ubuntu (ver Figura 16) y en Windows (ver Figura 17). En la parte derecha de cada imagen se muestra el Espacio Privado y en la parte izquierda se muestra el espacio Público.

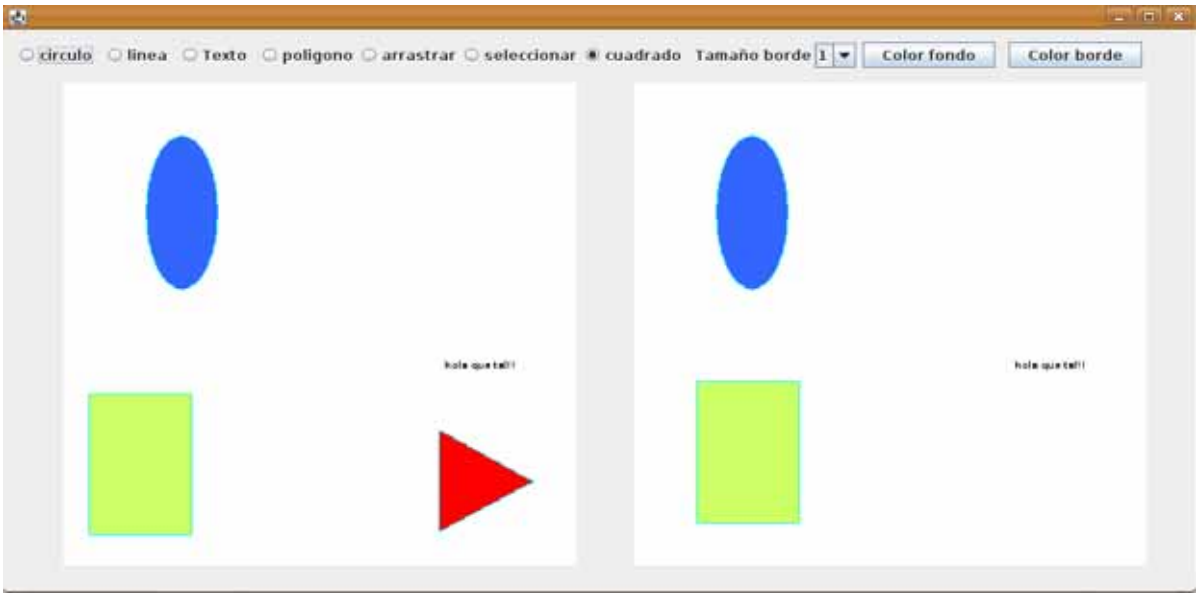


Figura 16: Aplicación corriendo en Ubuntu

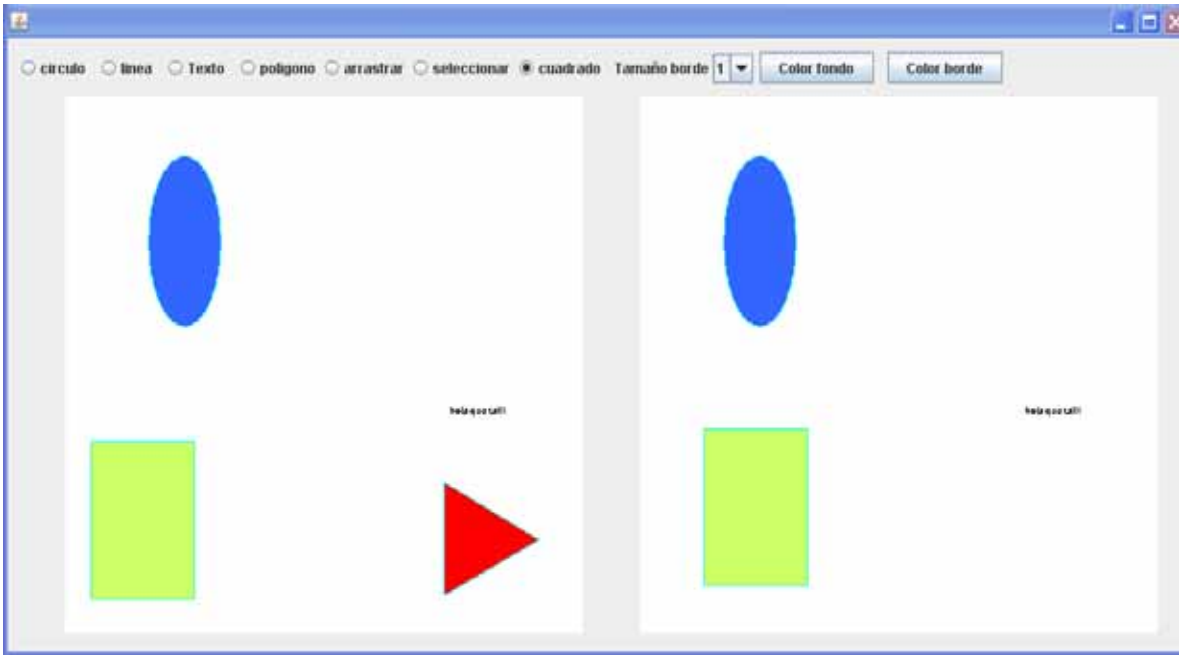


Figura 17: Aplicación corriendo en Windows

A continuación vamos a ver como se comparte una figura del espacio privado al espacio público y como es que se refleja el cambio en todos los clientes conectados al servidor. En el espacio de trabajo privado colocamos el cursor sobre una figura; luego, oprimimos el botón derecho y se despliega un menú; ahí elegimos la opción de compartir; y finalmente, se envía el objeto al espacio público, reflejándose los cambios en todos los clientes(ver Figura 18).

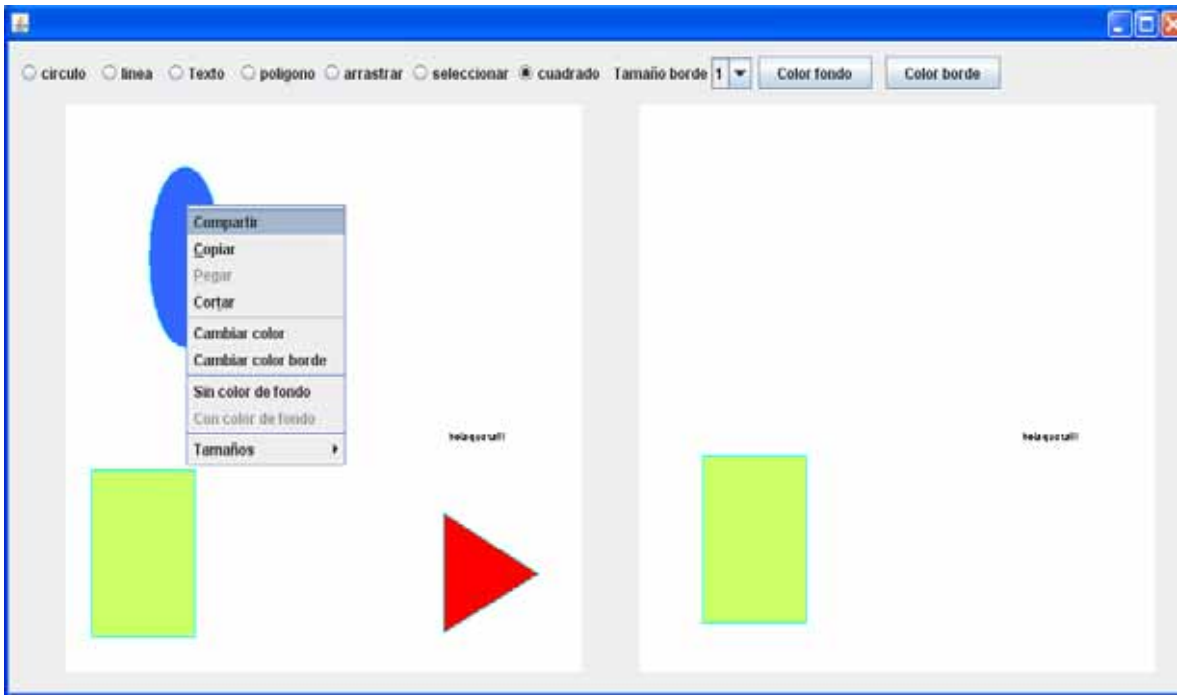


Figura 18: Compartir Objeto

Se comparte la figura en el espacio público del cliente que solicitó la acción (ver Figura 19).

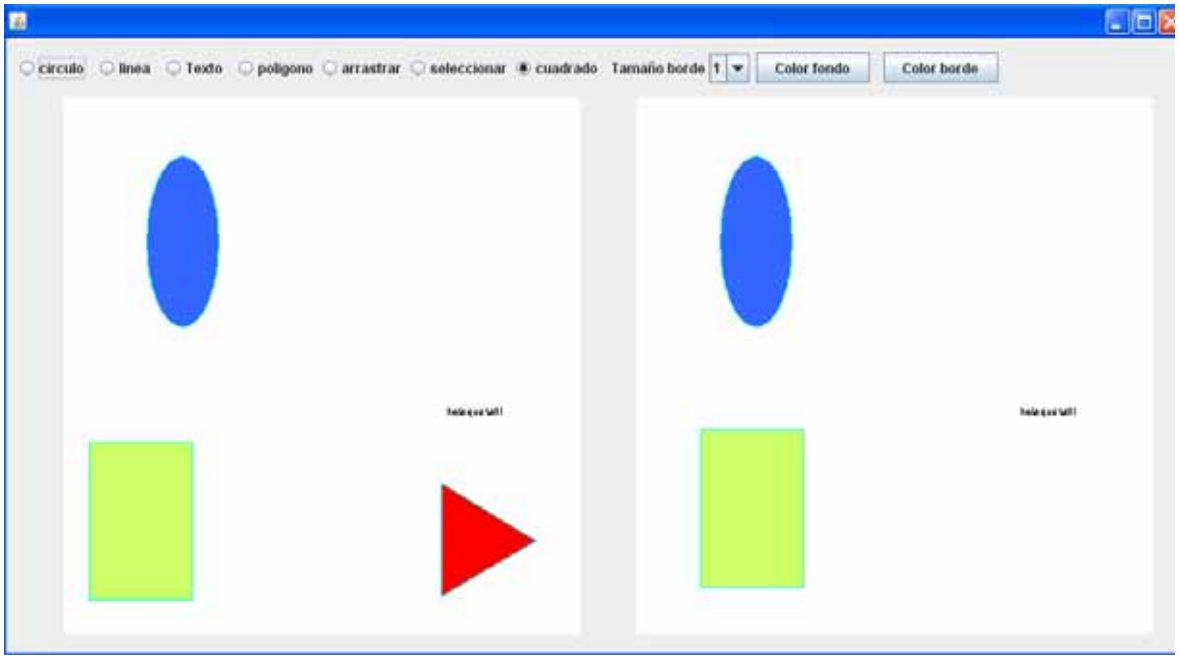


Figura 19: Objeto compartido en espacio Público

Se comparte el objeto en el espacio Público del cliente ejecutándose en Ubuntu (ver Figura 20).

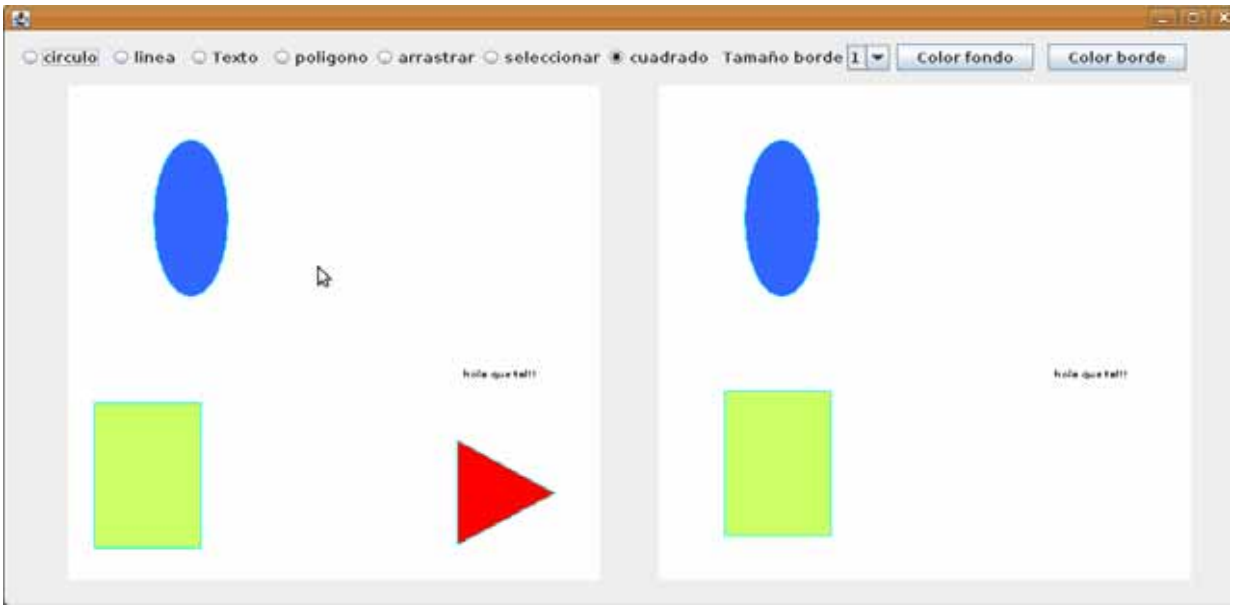


Figura 20: Objeto compartido en espacio Público

Ahora vamos a ver la funcionalidad de cortar una figura, colocamos el cursor en la figura y damos clic en el botón derecho del ratón, elegimos la opción de cortar(ver Figura 21).

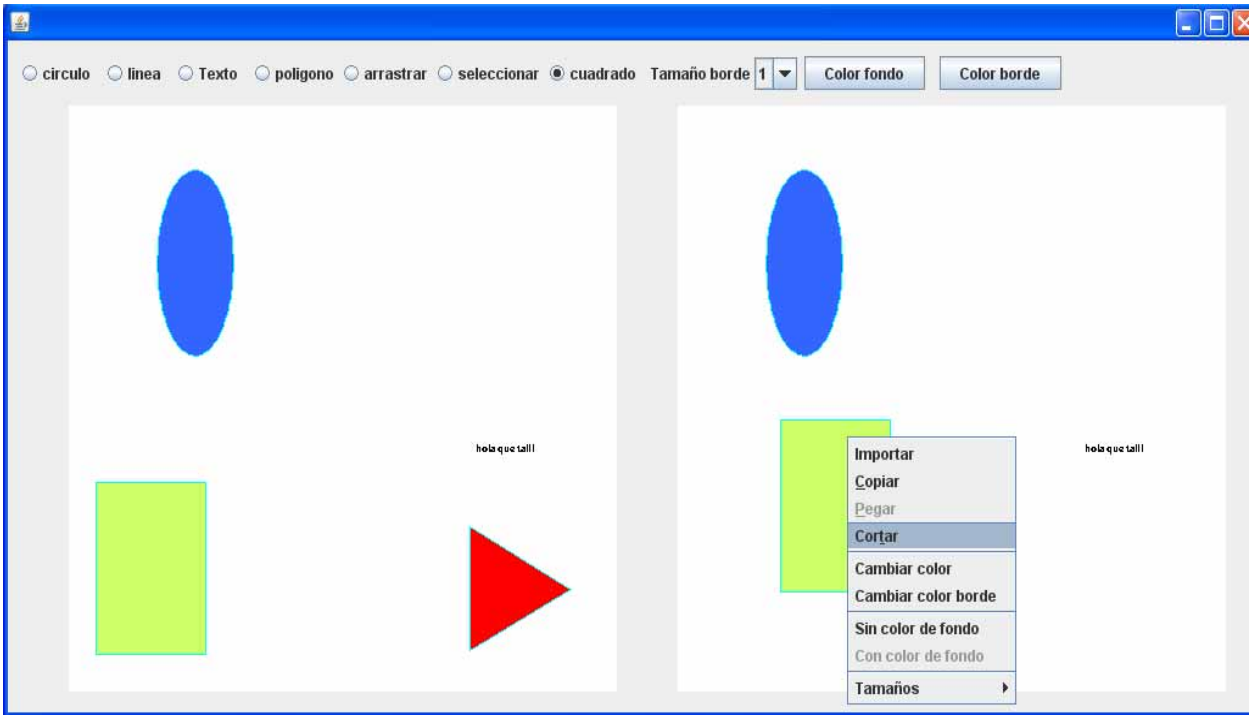


Figura 21: Opción de cortar

Los cambios se reflejan en ambas aplicaciones como se muestra en las Figura 22 y Figura 23).

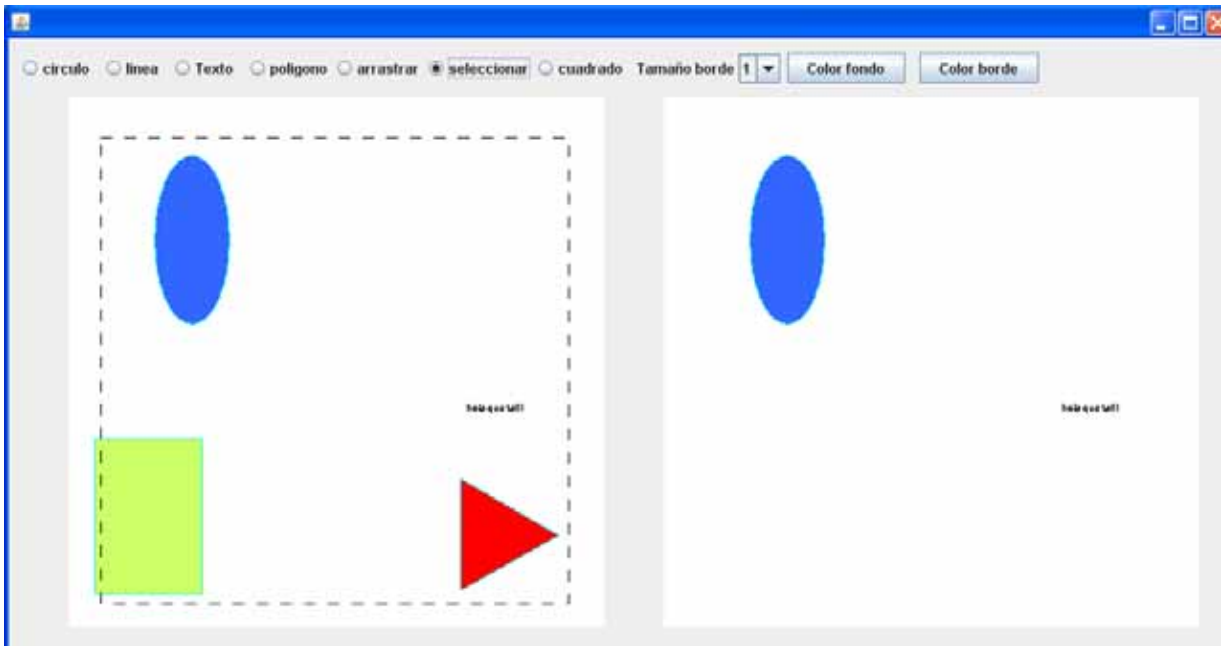


Figura 22: Objeto eliminado en Windows

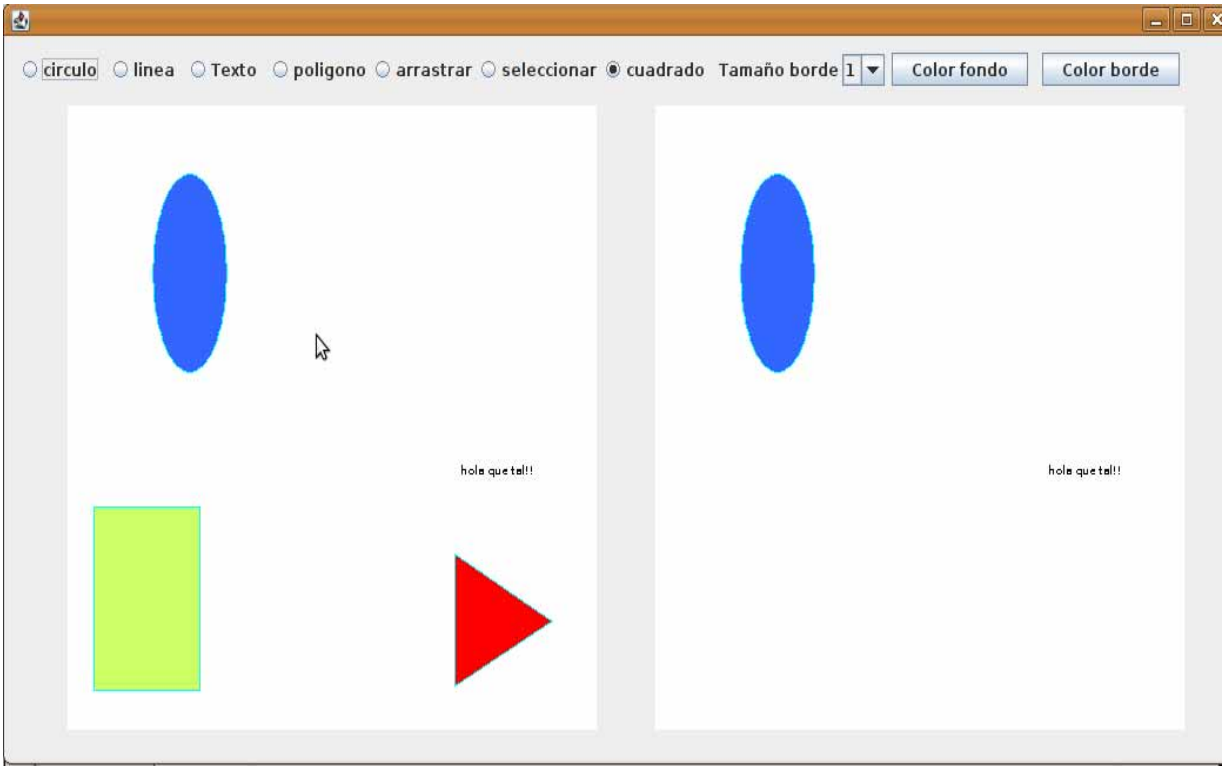


Figura 23: Objeto eliminado en Ubuntu

En la Figura 22 se muestra un cuadro punteado en el espacio privado, este cuadro aparece una vez elegida la opción de seleccionar en la barra de herramientas; podemos compartir esta selección o

eliminarla. Las opciones que disponemos se muestran en el menú desplegable que aparece al hacer clic derecho en el ratón.

La primera opción del menú (ver Figura 24) cambia dependiendo del espacio de trabajo donde nos encontremos, para el espacio de trabajo privado esta opción aparece como compartir; y para el espacio de trabajo público aparece como importar. Todas las demás opciones se aplican sobre uno o varios objetos seleccionados pero solo es el espacio de trabajo donde se encuentren dichos objetos.

Figura 24: Menú desplegable

Para finalizar con la demostración de la aplicación mostraremos hasta donde llegamos con el dispositivo móvil. Primero mostramos en la Figura 23 que la aplicación consta de los dos espacios pero esta vez no se encuentran divididos en el área del dispositivo, sino que se son dos pestañas en la parte superior; al colocar el puntero sobre alguna pestaña cambia de color para mostrarnos en que espacio estamos trabajando. Aquí no contamos con una barra de herramientas sino que las acciones suceden con el teclado del teléfono. Cabe aclarar que la aplicación se debe de montar sobre un dispositivo touchscreen para que al arrastrar el pointer sobre cualquier espacio se dibuje la figura.

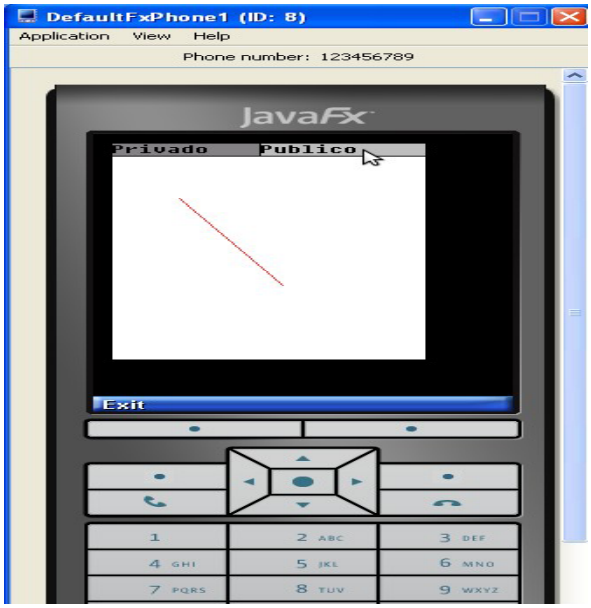


Figura 25: Aplicación ejecutándose en el dispositivo móvil

En la Figura 25 se muestra una línea dibujada en el espacio público. Para generar las pestañas de selección nos encontramos con muchos problemas ya que J2ME no proporciona un API para generar estas divisiones por lo que lo tuvimos que programar por pixel.

En la Figura 26 se muestra un círculo dibujado en el área privada.

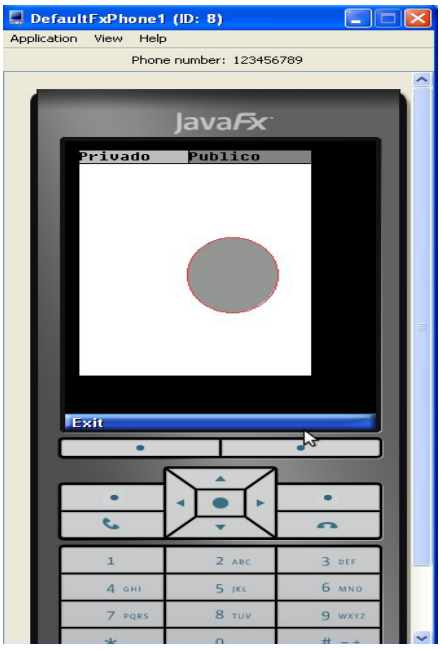


Figura 26: Dibujando un circulo

Y por último mostramos como se comparte el círculo que se encuentra en el espacio privado al espacio público (ver Figura 27), la manera de compartir objetos entre espacios es dando doble clic en la figura, cabe mencionar que tampoco existe el doble clic en J2ME por lo que tuvimos que solucionar esto midiendo el tiempo entre los clic del puntero, si el espacio de tiempo entre dos clic es muy pequeño lo tomamos como un doble clic.

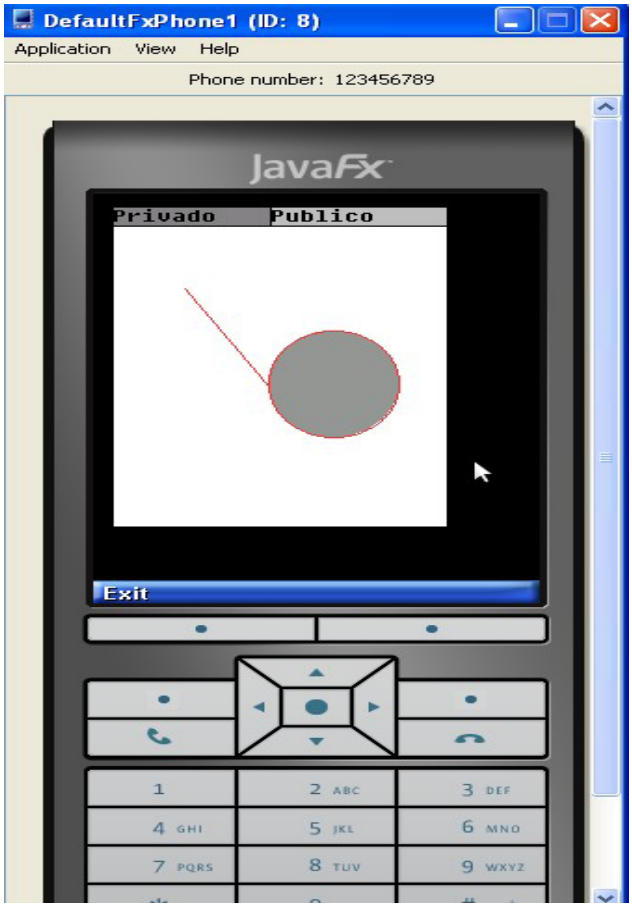


Figura 27: Circulo compartido

Conclusiones y trabajo futuro

Un pizarrón colaborativo es una aplicación de dibujo en proceso de desarrollo donde será posible crear y editar dibujos a partir de figuras geométricas básicas, y en donde varios usuarios podrán trabajar sobre la edición de estas figuras; cada uno de los usuarios podrá estar conectado a la aplicación a través de una red de dispositivos a saber: PC, laptop, PDA o teléfono. El presente proyecto tuvo como objetivo general el dotar a esta aplicación con los espacios público y privado donde los usuarios puedan realizar y compartir sus dibujos. La tabla 25 muestra los objetivos específicos, así como el alacance que se obtuvo con el desarrollo de este proyecto terminal.

	Objetivos	**Descripción**
✔	Diseñar un algoritmo que proporcione la propiedad de plasticidad para los espacios de trabajo público y privado de la aplicación “pizarrón compartido”.	Cumplimos con el diseño del algoritmo que permite compartir objetos entre los espacios público y privado.
✽	Implementar el algoritmo que proporcione la propiedad de plasticidad para los espacios de trabajo público y privado.	La implementación del algoritmo que proporciona los espacios público-privado para PC se alcanzó totalmente. La implementación del mismo algoritmo para dispositivo móvil no se concluyó completamente, porque se requiere implementar clases de JAVA estándar en J2ME; y el tiempo del proyecto se agotó.
✽	Realizar pruebas sobre los espacios público y privado, con la propiedad de plasticidad, en un dispositivo PDA y en una PC.	Las pruebas en PC se realizarón tanto a nivel de conexión, como a nivel de manipulación de los espacios público y privado. Sin embargo, para dispositivos móviles no se alcanzó este objetivo, ya que no se termino de implementar la comunicación con los dispositivos, esto debido al tiempo para realizar el proyecto.
✔	Elaborar manuales de usuario y manuales de instalación, de los espacios público y privado.	
✔	Elaborar el reporte final.	
✔✔	Permitir más de dos clientes (dibujantes) conectados al servidor en un mismo instante.	En la aplicación anterior solo se permitían dos clientes conectados y compartiendo el dibujo en el mismo instante. En este proyecto se implementó un servidor con sockets que permite que más de dos clientes se conecten en un mismo instante.
✔✔	Las figuras (polígono, rectángulo y círculo) pueden tener color de relleno.	Se diseño e implemento la aplicación para que las figuras creadas puedan tener color de relleno, además de seleccionar el color de relleno.
✔✔	Seleccionar más de un objeto para poder compartirlo, eliminarlo o copiarlo.	Se elaboró un cuadro de selección donde el usuario puede elegir más de un objeto al mismo tiempo.
✔✔	Un objeto tiene funcionalidad de modificar el tamaño del borde antes y después de crear dicho objeto.	A partir de un menú desplegable el usuario puede modificar el tamaño de borde de un objeto, ésto incluye al rectángulo, la elipse y el polígono.
✔✔	Persistencia de los datos	Se utilizó XML para guardar los cambios efectuados durante una sesión previa.
✔✔	Transmisión de los datos	En un principio se pensó utilizar los objetos serializables para poder realizar la transmisión de los mismos a través de la red. Pero nos encontramos con dificultades al momento de implementar la comunicación con los objetos serializables ya que

		no llegaban los mensajes correctamente de un equipo a otro, por lo que fue mas difícil pasar objetos serializables a través de internet que pasar una cadena de texto con que represente a un objeto, para realizar esta operación, por lo que decidimos crear clases que se encargaran de convertir un objeto tipo Figura, Texto y Polígono a una representación XML, y viceversa.

Tabla 25: Objetivos alcanzados

✽ Objetivo alcanzado parcialmente.
✔ Objetivo alcanzado.
✔✔ Objetivo alcanzado no propuesto (objetivos extras).

La realización de este proyecto nos permitió, aprender más sobre el lenguaje JAVA, además de aprender nuevas tecnologías, tal como lo es XML; también profundizamos sobre la comunicación con Sockets; además, encontramos que en los dispositivos móviles ya casi no se emplea J2ME, esto se debe a que cada fabricante de dispositivos móviles proporciona su propia API de programación, y aunque esta API esté hecha en JAVA existe incompatibilidad con los API de otros dispositivos móviles.

Consideramos que el avance sobre la aplicación “pizarrón compartido” fue sustancial; sin embargo, aún hay funcionalidades por desarrollar, a saber:

- Sincronizar, al momento que un cliente se conecte al servidor; para que obtenga la última versión del dibujo creado en el espacio público.
- Concluir la conexión con dispositivos móviles.
- Guardar automáticamente los cambios realizados en los dibujos durante la sesión de trabajo.
- Implementar la funcionalidad de *drag and drop* (arrastrar y soltar) para compartir las figuras entre los espacios de trabajo público y privado.

Bibliografía

[1] Clarence A. Ellis, Simon J. Gibbs, Gail L. Rein: “*Groupware: Some Issues and Experiences. Commun"*. ACM 34(1): 39-58 (1991)

[2] Saul Greenberg, Michael Boyle, Jason C. Laberge: “PDAs and Shared Public Displays: Making Personal Information Public, and Public Information Personal”. *Personal and Ubiquitous Computing* 3(1/2): (1999)

[3] Gabriela Sánchez Morales: Redistribución semi-plástica mixta: el caso de estudio de un pizarrón compartido. Tesis de Maestría CINVESTAV IPN, febrero de 2009

[4] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html

[5] Harvey M. Deitel Paul J. Deitel: “Cómo programar en JAVA”. Pearson Educación: Quinta Edición (2004)