**CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL**

**UNIDAD ZACATENCO**

**DEPARTAMENTO DE COMPUTACIÓN**

# OBJETOS DISTRIBUIDOS VISUALES

## TESIS

Que presenta

LAURA PATRICIA RAMÍREZ RIVERA

Para obtener el grado de

## DOCTORA EN CIENCIAS EN COMPUTACIÓN

Codirectores de Tesis:

**Sergio Víctor Chapa Vergara**

**Amilcar Meneses Viveros**

**México, D.F.** **November 2013**

CENTRO DE INVESTIGACION Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITECNICO NACIONAL

**UNIDAD ZACATENCO**

**DEPARTAMENTO DE COMPUTACION**

# DISTRIBUTED VISUAL OBJECT

**A dissertation submitted by:**

LAURA PATRICIA RAMIREZ RIVERA

**for the degree of**

DOCTOR OF COMPUTER SCIENCE

**Supervisors:**

Sergio Víctor Chapa Vergara

Amilcar Meneses Viveros

**México, D.F.** **November 2013**

# Agradecimientos

Quiero agradecer a Conacyt por brindarme la oportunidad de obtener este grado académico, ya que sin su apoyo económico no me hubiera sido posible realizar esta investigación. Además me gustaría agradecer al CINVESTAV su apoyo constante durante mi estancia en sus instalaciones. A mis asesores que me brindaron su confianza y me permitieron explotar mis capacidades de recopilación y análisis, dando como resultado el trabajo final que en esta tesis se presenta. Quiero agradecer a mi padres y a mi hija que han sido mi apoyo incondicional durante este proceso.

# Índice general

# Índice de figuras

# Lista de acrónimos

- **API** Application Programming Interface (Interfaz de programación de aplicaciones)
- **CORBA** Common Object Request Broker Architecture (Arquitectura común de broker de peticiones de objetos)
- **DMX** Distributed Multi-head X server
- **DVO** Distributed Visual Objects (Objetos distribuidos visuales)
- **GB** GygaBytes
- **GLUT** OpenGL Utility Toolkit (Conjunto de herramientas de OpenGL)
- **GPU** Graphics Processing Unit (Unidad de procesamiento gráfico)
- **GUI** Graphical user interface (Interfaz gráfica de usuario)
- **HCI** Human Computer Interaction (Interacción Humano-Computadora)
- **IP** Internet Protocol (Protocolo de internet)
- **JVM** Java Virtual Machine (Máquina virtual de java)
- **LCD** Liquid Crystal Display (Pantalla de cristal líquido)
- **LED** Light-Emitting Diode (Diodo de emisión de luz)
- **LOD** Limit Of Detection (Minima cantidad de substancia que puede ser distinguida)
- **MPI** Message Passing Interface (Interfaz de paso de mensajes)
- **MVC** Model-View-Control (Patrón de diseño Modelo-vista-controlador)
- **OMA** Object Management Architecture (Arquitectura de manejo de objetos)
- **OSG** Open Scene Graphic (Herramientas de gráficos 3D basada en OpenGL )

- **OSI** Open System Interconnection (Sistema de interconexión abierto)
- **OpenGL** Open Graphic Library (Biblioteca de gráfica abierta)
- **POO** Programación Orientada a Objetos
- **Pes** Processing Elements (Elementos de procesamiento)
- **RFB** Remote Frame Buffer (memoria gráfica remota)
- **RGB** Red Green Blue (Componentes de color en base a la combinación de rojo,verde y azul)
- **RISC** Reduced Instruction Set Computer (Conjunto de instrucciones de computadora reducido)
- **RMI** Remote Method Invocation (Invocación remota de método)
- **SAGE** Scalable Adaptive Graphics Environment (Ambiente escalable adaptativo para gráficos)
- **SDL** Simple DirectMedia Layer (Biblioteca multimedia para acceso de bajo nivel a dispositivos de entrada)
- **SLI** Scalable Link Interface (Liga de interfaz escalable)
- **TCP** Transport Control Protocol (Protocolo de control de transporte)
- **UDP** User Datagram Protocol (Protocolo de datagrama de usuario)
- **UML** Unified Modeling Language (Lenguaje de modelado unificado)
- **VGA** Video Graphics Array (Arreglo gráfico de vídeo)
- **VNC** Virtual Network Computing (Computación Virtual en Red)
- **VO** Visual Object (Objeto visual)

# Resumen

Las paredes de vídeo son una herramienta de la visualización científica que permite el despliegue de visualizaciones en alta resolución y en gran tamaño. Una de las ventajas proporcionadas por esta herramienta es facilita al usuario o usuarios la comprensión de grandes cantidades de datos numéricos, ya que son convertidos en visualizaciones que son más fáciles de entender. Además su gran tamaño genera un ambiente envolvente que mejora la atención del usuario.

La mayoría de las aplicaciones de paredes de vídeo son creadas para resolver problemas específicos, algunas otras trabajan mediante entornos de aplicaciones. Haciendo un análisis de cada investigación es posible obtener una abstracción jerárquica del problema, que permita identificar los principales componentes necesarios en el desarrollo de este tipo de aplicaciones; el desarrollo de este tipo de aplicaciones es complicado ya que involucra diversos aspectos (visualización, redes, interacción hombre-máquina, etc.). De modo que hasta ahora no existe un estándar que facilite la generación de este tipo de aplicaciones.

De los componentes principales obtenidos en esta abstracción se obtienen las funciones generalizadas, estas funciones generalizadas pueden ser identificadas independientemente de la aplicación final. En base a las funciones generalizadas es posible desarrollar un modelo que permita la creación de diversos tipos de aplicaciones.

El uso de este modelo puede ser empleado en la construcción de un manejador de ventanas distribuido. Este manejador facilitará el despliegue de cualquier tipo de visualización en los ambientes de paredes de video. En esta tesis se propone un modelo basado en el paradigma orientado a objetos que permitirá la creación de un manejador de ventanas distribuido.

Este modelo esta basado en capas, donde cada capa se encarga de realizar cada una de las funciones generalizadas. Este modelo se es propuesto como base para la creación de un manejador de ventanas distribuido. Este manejador de ventanas distribuido será el componente principal en la creación de un escritorio remoto distribuido, facilitando la generación de aplicaciones para la pared de vídeo. El componente principal del modelo es el objeto distribuido visual, el cual tiene su origen en la composición de los objetos distribuidos y visuales.

**Keywords:**Visualización científica, pared de vídeo, objetos distribuidos, objetos visuales, objetos distribuidos visuales

# Abstract

Because large scale display provides advantages to obtain knowledge from a big amount of data, they increase the use in many scientific areas. This kind of display lets the visual representation of numeric data in high resolution and big size. One advantage of the large scale display is the environment generated for the user, this environment lets the user to obtain a better knowledge of the visual data representation.

There are many large scale display applications, most of them are made to solve specific problems. We can make an analyze of these approaches to obtain an abstraction of the problem. This abstraction lets get the main components required in the development of this kind of applications; because the requirements of this kind of applications are many in different areas such as visualization, networking, human-computer interaction, the building of this kind of applications is hard. Actually any standard has been made to ease the building of this kind of applications. A model to ease the building of this kind of applications is required.

The main components obtained from the abstraction are the general functions, these functions can be made independently of the final application. Based on the general functions is possible get a model to let building different kinds of applications.

This model is the main component to develop a distributed window manager. This manager will ease the display of whatever kind of visualization in large scale display environment. In this thesis this model is proposed object-oriented based.

The model is divided in layers, each layer implements one general function. The principal component of the model is the distributed visual object, which is defined by the composition of the distributed object and visual object.

**Keywords:** scientific visualization, large scale display, distributed object, visual object, distributed visual object.

# Introducción

## Antecedentes

Este proyecto inicio en el año 2006, cuando se adquirió el hardware necesario para construir una pared de vídeo de alta resolución. La construcción de lo que hoy se conoce como CinvesWall permitió la generación de aplicaciones que aceptan el despliegue de visualizaciones en alta resolución. Una de las primeras aplicaciones desarrolladas en el CinvesWall fue un Minero de Datos Visual (MDV), elaborado con la finalidad de obtener datos de diversas fuentes y generar la correlación de Pearson. Este minero puede desplegar múltiples ventanas conteniendo esferas en 3D con información relativa a la media, varianza y desviación estándar del conjunto de datos de entrada. Estas ventanas se controlan mediante el manejo del ratón (ver figura 1 ).

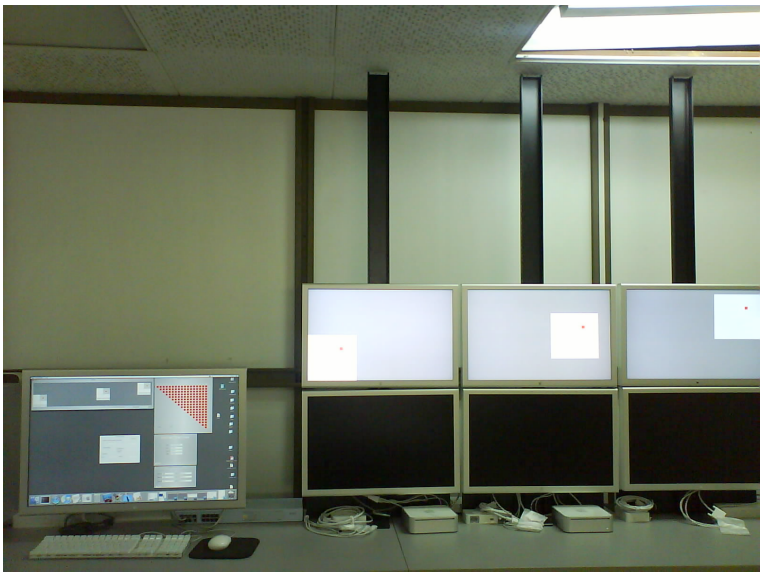

Figura 1: Vista del Minero de Datos Visual en el CinvesWall

Durante el proceso de diseño del MDV se encontraron diversos problemas con el uso

de red y la distribución de datos. Estos mismos problemas se encontraron en aplicaciones con funcionalidad distinta, tal como el visor de imágenes y el reproductor de vídeo. Los problemas encontrados y la variedad de aplicaciones por desarrollar nos dirigieron a la búsqueda de un componente general que permitiera el uso de cualquier tipo de aplicación sobre la pared de vídeo.

Este componente debe cumplir con características de distribución y visualización suficientes para el desarrollo de aplicaciones en ambientes de paredes de vídeo. La búsqueda de este componente es el antecedente principal que nos impulsó a iniciar esta investigación.

## Paredes de vídeo en la visualización científica

La visualización científica es un área de la computación que surgió como auxiliar para facilitar la obtención del conocimiento dentro de grandes cantidades de datos numéricos. Esta área mejora el entendimiento de problemas complejos a través de sus métodos y tecnologías (ver figura 2.3). En los problemas científicos permite exponer los resultados de la experimentación computacional gráficamente[Wilson, 1998]. Es por eso que su uso se ha extendido en diversas áreas de la investigación[Renambot et al., 2008].

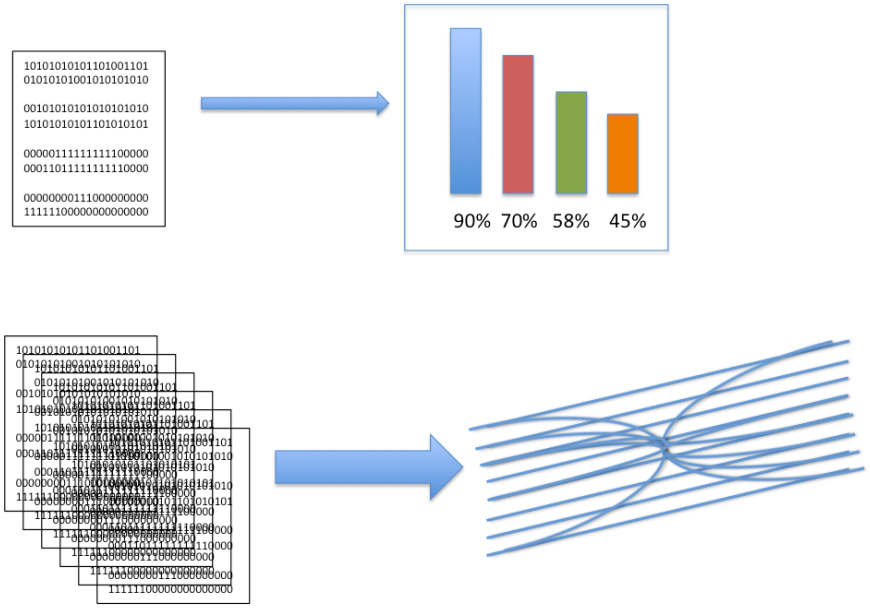


Figura 2: Gráficos empleados en la visualización científica

Un concepto muy importante en la visualización científica es la percepción de la información. La percepción de la información es la facilidad con la que los científicos pueden entender los resultados gráficos de la visualización científica. Uno de los puntos fundamentales de la percepción de la información establece, que existe una correspondencia entre el

tipo de gráfico generado y el dispositivo de salida. Dependiendo del tamaño del conjunto de datos graficados, la escalabilidad empieza a ser un factor importante. Es por eso que las visualizaciones científicas se implementan sobre gráficos escalables, que puedan ser desplegadas en áreas de grandes dimensiones[Kurtenbach and Fitzmaurice, 2005].

Existen diversos tipos de despliegues utilizados en la ciencia, algunos proporcionan el despliegue en alta resolución o en gran tamaño (ver figura 3). Particularmente, las paredes de vídeo permiten desplegar grandes volúmenes de información en alta resolución con un gran tamaño[Leigh et al., 2006].

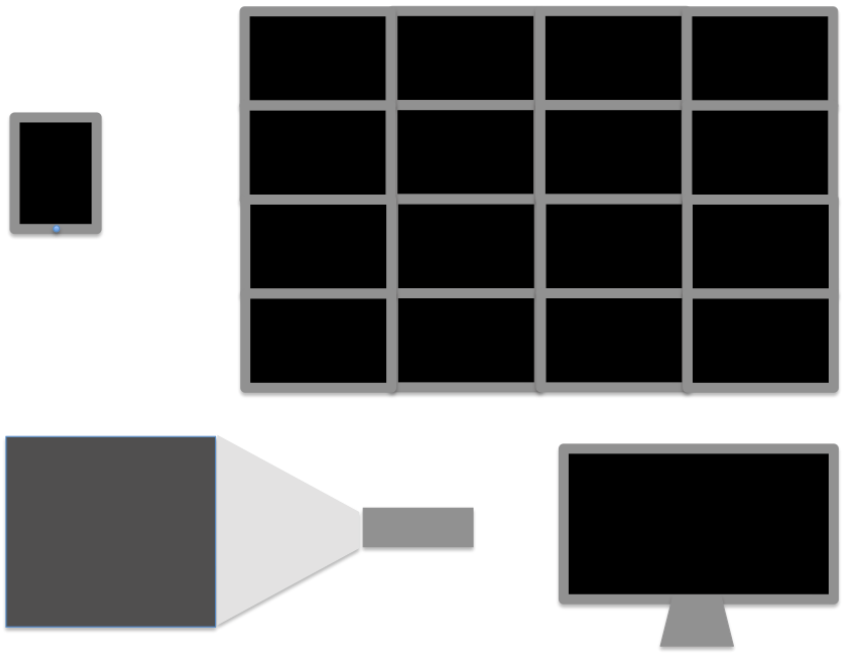

Figura 3: Herramientas de la visualización científica

Diversos trabajos han demostrado los importantes beneficios obtenidos en la productividad de los científicos que emplean paredes de vídeo. Esto se debe gracias a sus características de alta resolución y gran tamaño que mejoran la percepción de la información de los científicos.

Una de las características que mejora la percepción de la información es la gran dimensión que tiene la pared de vídeo. Esta característica genera un ambiente envolvente para el científico que lo hace sentir inmerso en el despliegue (ver figura 4).

Debido al gran tamaño de la pared de video, se genera un amplio espacio frente a la pared de vídeo, esto permite a múltiples usuarios una vista simultánea del desplie-

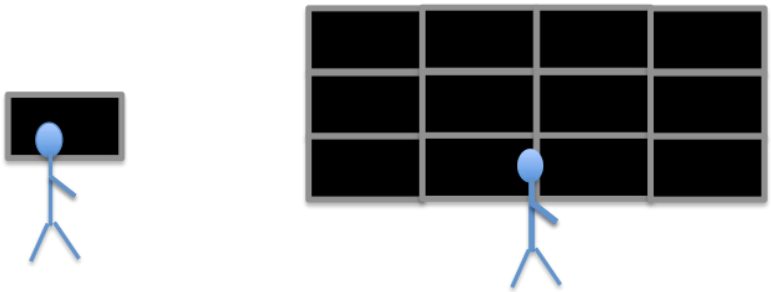

Figura 4: La pared de video genera un ambiente envolvente para el científico

gue (ver figura 5). De esta manera esta herramienta puede ser aprovechada por diversos científicos al mismo tiempo permitiendo la comparación de perspectivas dado un mismo despliegue[Funkhouser and Li, 2000].

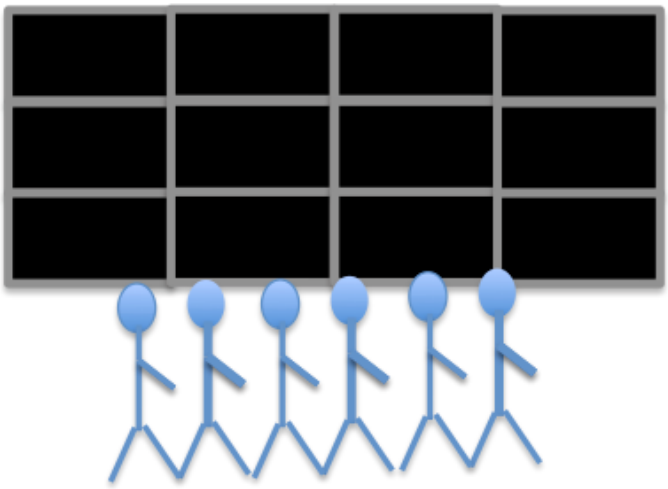

Figura 5: Múltiples usuarios tienen una vista simultánea del despliegue

Debido a que la pared de vídeo proporciona alta resolución y gran tamaño permite mostrar los detalles del conjunto de los datos desplegados, así como diversas perspectivas de los mismos datos. De ese modo es posible hacer comparaciones con múltiples perspectivas de la misma información y desplegarlas al mismo tiempo (ver figura 6).

Todas estas características hacen que la pared de vídeo sea una de las mejores herramientas existentes para la visualización científica, de ahí nuestro interés en conocer y explicar cómo es su manejo.

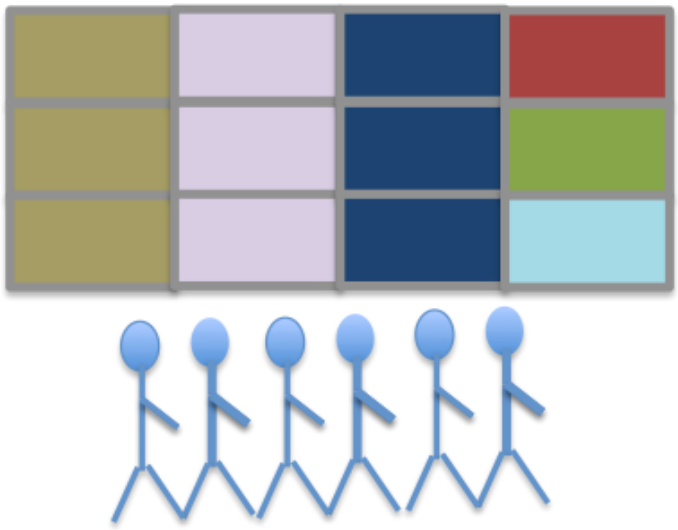

Figura 6: Diferentes perspectivas de la información pueden ser mostradas

Durante la evolución de las paredes de vídeo han ocurrido diversos cambios en su tecnología. De modo que le han permitido mejorar su desempeño (procesamiento y despliegue). De tal manera que su denominación ha variado (video wall, large display, tiled display o large scale display). Dados estos problemas de traducción hemos optado por mantener el termino pared de vídeo (video wall) y definirlo como:

*Una pared de vídeo es un conjunto de pantallas controladas de alguna manera, con la finalidad de obtener un despliegue de grandes dimensiones con alta resolución.*

## Tecnologías de despliegue de la pared de vídeo

Debido a las características mencionadas con anterioridad sabemos que el despliegue de las paredes de vídeo debe cumplir con dos aspectos principales: (1) alta resolución y (2) grandes dimensiones. Es por eso que la obtención de un despliegue de este tipo requiere procesos especiales como procesamiento para la generación de la visualización, conectividad para enviar el despliegue generado o parte de él y espacio en la memoria para almacenar la matriz de pixeles formada.

Estas restricciones requieren el uso de tecnologías específicas encargadas de la generación del despliegue.

De esta necesidad surgen dos enfoques principales: (1) Estación de visualización; es un arreglo de tarjetas gráficas conectadas en cascada encargadas de generar una visualización en común y (2) Cluster de visualización; es un conjunto de máquinas encargadas de

generar una visualización en común. Cada una de ellas tiene ventajas y desventajas que se muestran a continuación (ver figura 7)

| Características | Cluster de visualización | Estación de visualización |
|---|---|---|
| Costo | Regular | Elevado |
| Fácil de programar | No | Si |
| Desempeño | Programable | Promedio |
| Escalabilidad | Ilimitada | Limitada |

Figura 7: Comparación de las tecnologías utilizadas en el despliegue de paredes de vídeo

La estación de visualización esta formada principalmente por tarjetas gráficas de alto desempeño por lo que su costo es elevado. Cada tarjeta maneja una o dos pantallas dependiendo de la configuración, de modo que tener una pared de vídeo de este tipo es costoso. Además en este caso el número de pantallas esta limitada, después de este limite el desempeño empieza a disminuir. Su principal ventaja es que esta basado en hardware y su implementación es casi inmediata.

El cluster de visualización es de menor costo que la estación de visualización, su escalabilidad es ilimitada ya que se emplea un sistema distribuido para su control. Esto permite incrementar el número de pantallas sin generar perdida en el desempeño. El principal inconveniente de este tipo de paredes es que el software es difícil de programar ya que se requiere un sistema distribuido el cual implica envío de mensajes, sincronización y distribución de la información.

Ambas tecnologías son ampliamente usadas, pero el cluster de visualización es el que proporciona mayor flexibilidad en la programación del manejo de la pared de vídeo y en la generación del gráfico. Es por eso que en este proyecto se trabaja con la tecnología basada en cluster de visualización.

El cluster de visualización permite una mayor flexibilidad en los componentes utilizados en la arquitectura de la pared de vídeo (ver figura 8). Esta arquitectura se puede dividir en 3 bloques principales: (1) servidor de visualización, (2) cluster de visualización y (3) bloque de monitores. El servidor de visualización es el encargado de enviar la información al cluster de visualización. El cluster de visualización se encarga de desplegar la información en las pantallas.

El servidor de visualización en ocasiones esta incluido dentro de los nodos del cluster de visualización, dependiendo del modelo lógico empleado. Este servidor es el encargado del manejo de la interfaz gráfica de usuario (Graphical User Interface, GUI).

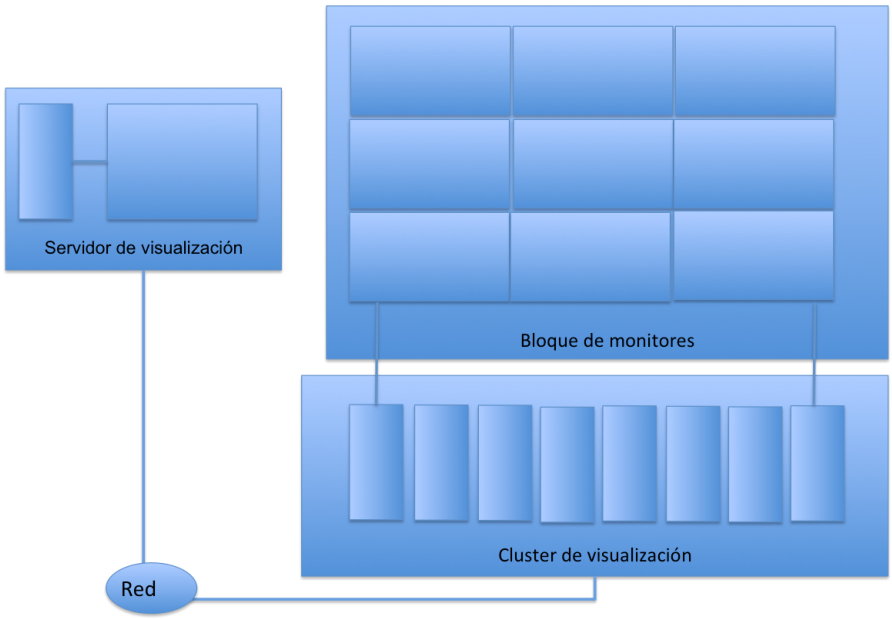


Figura 8: Arquitectura más común en hardware de una pared de vídeo

Basados en el manejo de la pared de vídeo mediante el cluster de visualización, es posible elegir algún tipo de modelo lógico de comportamiento, a continuación se describen 3 de los modelos más sobresalientes[Chen et al., 2001b][Chen et al., 2001a]:

- *Modelo Cliente-Servidor*.- En este modelo el control total se mantiene en el cliente, el cliente se encarga de la interacción con el usuario y de la distribución de los datos y el cliente emplea a los servidores para desplegar la información en las pantallas (ver figura 9).

  En el cliente se encuentra la interfaz gráfica de usuario, por lo que cuando los eventos son reconocidos deben ser traducidos y enviados a los servidores. De ese modo se replica la acción del usuario desde el cliente hasta los servidores. Cuando una visualización es desplegada en la pantalla del cliente, esta visualización es escalada y duplicada en las pantallas de la pared de vídeo.

  Este modelo permite dos modalidades de despliegue: (1) modo inmediato y (2) modo retenido. El modo inmediato indica que cuando la información del cliente cambia, el servidor recibe el mensaje de actualización y realiza el despliegue inmediatamente. El modo retenido indica que cuando la información del cliente cambia, el servidor recibe el mensaje lo compara con anteriores y lo almacena, de modo que puede reutilizar la información almacenada con anterioridad. Esto ocurre sobre todo cuando la actualización implica un cambio mínimo en la visualización desplegada.

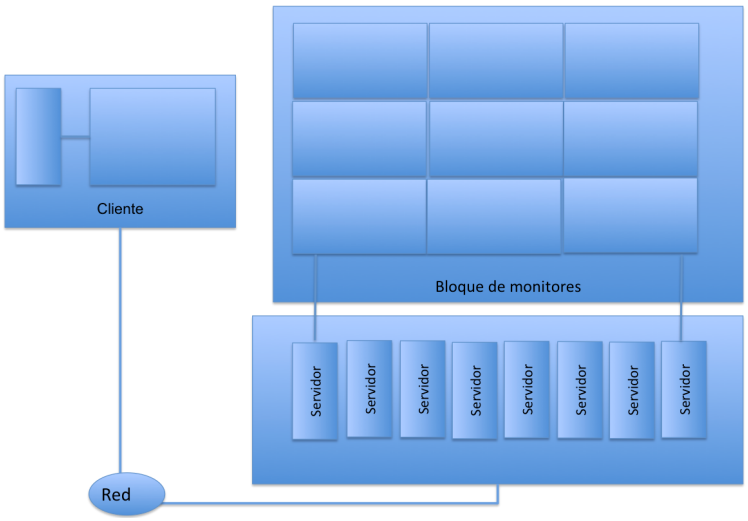


Figura 9: Modelo Cliente-Servidor

- *Modelo Maestro-Esclavo*.- Este modelo esta basado en la relación de jerarquía que se genera entre los nodos del cluster de visualización. De modo que el maestro es un nodo del cluster de visualización y además es el encargado de mantener el control de los demás nodos (ver figura 10).

  El nodo maestro de desplegar la interfaz gráfica de usuario y la parte de despliegue que tenga asignado. Este nodo es el encargado de manejar los eventos de usuario y replicarlos a los nodos esclavos.

  Este modelo también permite las dos modalidades de despliegue mencionadas en el modelo cliente-servidor. La principal diferencia entre los dos modelos es la existencia de un servidor de visualización independiente en el modelo cliente-servidor.

- *Modelo de ejecución sincronizada*.- En este modelo se realizan copias de una aplicación que se ejecuta en cada nodo del cluster de visualización (ver figura 12). Estas aplicaciones deben permanecer sincronizadas y coordinadas automáticamente dando la apariencia de ser una sola aplicación. En este modelo la información a desplegar se obtiene desde cada nodo y la interacción con el usuario se realiza directamente desde la pared de vídeo.

Los modelos lógicos mencionados anteriormente permiten diferentes niveles de control en cuanto al manejo de la GUI.

El modelo cliente-servidor permite la interacción con el usuario mediante el servidor de visualizaciones ubicado en el cliente. De esta manera es posible hacer la distribución de los datos a desplegar. Además el tamaño de la interfaz de usuario se mantiene con las dimensiones empleadas en un escritorio estándar.

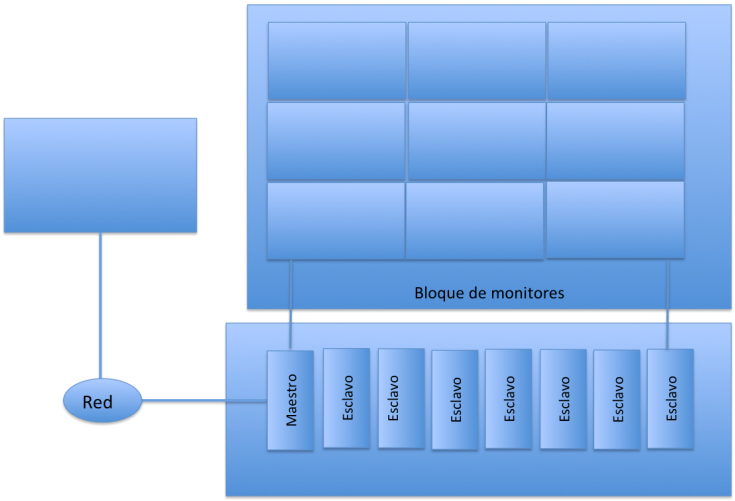


Figura 10: Modelo Maestro-Esclavo

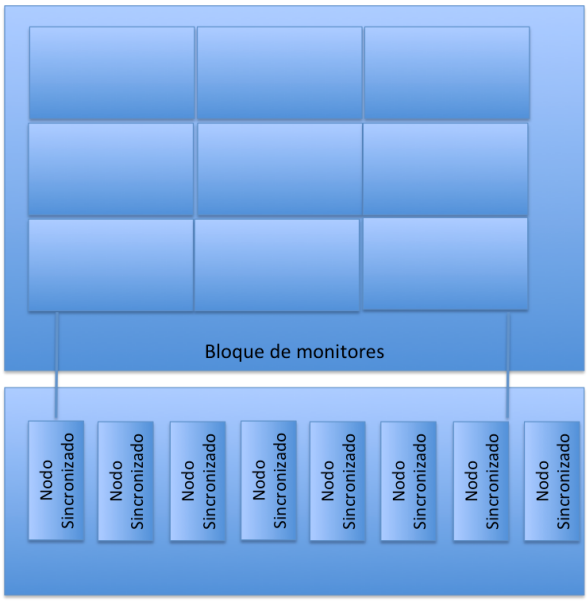


Figura 11: Modelo de ejecución sincronizada

El modelo maestro-esclavo proporciona una interacción con la pared de vídeo completa, por lo que el manejo de la interfaz de usuario cambia con respecto a un escritorio estándar y puede ser un poco más incomodo de usar.

El modelo de ejecución sincronizada distribuye la aplicación de forma que cada nodo del cluster realice sus funciones de manera independiente. Para lograrlo es necesario mantener una fuente de información continua. De modo que la aplicación pueda ejecutarse

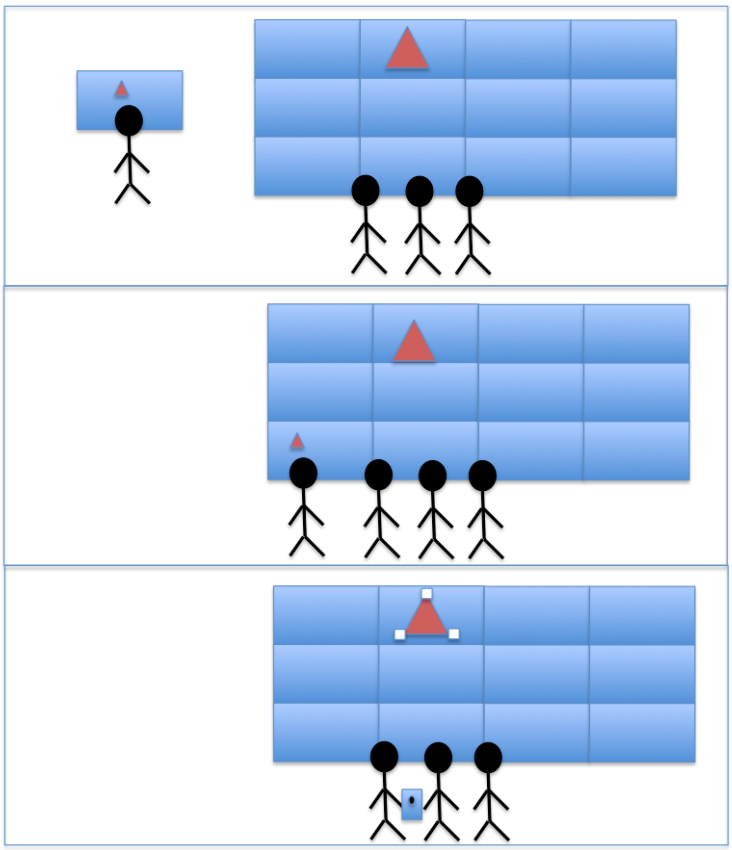

Figura 12: Niveles de control de la GUI

automáticamente y la interacción con el usuario sea directamente con la pared de vídeo.

En conclusión debido a como funcionan los niveles de control, es el modelo cliente-servidor el que proporciona una GUI más familiar a los científicos. Esto se debe a que este modelo mantiene el control del despliegue al tamaño de un escritorio estándar. Es por eso que en esta tesis adoptaremos el modelo cliente-servidor para definir el control de la pared de vídeo.

## Planteamiento del problema

Las paredes de vídeo despliegan información en grandes dimensiones con alta resolución utilizando múltiples aplicaciones. El diseño e implementación de este tipo de aplicaciones es complicado ya que involucran múltiples aspectos ( redes, graficación, etc. ). Diversos proyectos han estado trabajando para facilitar la construcción de este tipo de aplicaciones, a partir de ellos se han obtenido middlewares y frameworks que tratan de disminuir esta complejidad. Estos proyectos pueden ser clasificados con base en su comportamiento principal en dos grupos:

- Render Paralelo
- Frame Buffer Compartido

En necesario aclarar que aun entre los miembros de un mismo grupo existen características muy particulares que los diferencian. Por ejemplo, de la biblioteca OpenGL, se generó una primera aproximación por parte de un grupo de desarrollo de Stanford University Computer para realizar el render paralelo, extendiendo las instrucciones definidas por default en esta biblioteca con MPI, este proyecto se llamó WireGL [Humphreys et al., 2001], el mismo grupo continuó investigando y mejorando su proyecto agregando funcionalidad para la distribución de los datos, de ahí surge Chromium [Humphreys et al., 2002]. Al modificar aspectos de protocolos de red surge CGLX [Doerr and Kuester, 2011] del Center of Graphics, visualization and Virtual Reality (GRAVITY), California Institute for Telecommunications and Information Technology(Calit2) y University of California. Además CGLX es ya un middleware a diferencia de los anteriores, pero la idea principal en todos sigue siendo el render paralelo, en el siguiente capítulo se hablara más a detalle de estos proyectos.

El Render Paralelo esta pensado para la generación de grandes modelos 3D, la idea principal es la distribución de las primitivas básicas de dibujo del modelo, de manera que cada nodo se encargue de una fracción del modelo. En este caso se aprovechan las ventajas de las bibliotecas de aceleración gráfica como es OpenGL, Quartz ó Open Scene Graph, unidas con algún tipo de envío de mensajes como MPI, UDP, TCP/IP, etc. Al enviar solo primitivas básicas el paquete no es tan grande y puede ser recibido adecuadamente. En este caso cada nodo se encarga de una fracción del despliegue y esta fracción no puede ser modificada en tiempo de ejecución y debe definirse en la configuración inicial.

El Frame Buffer Compartido, es un protocolo que envía la salida de la memoria gráfica desde un dispositivo hacia la pared de vídeo. Al igual que en el caso anterior podemos encontrar proyectos con características diversas que emplean el mismo principio. Por ejemplo el protocolo original llamado RFB [Richardson, 2010], toma la matriz de pixeles que contiene la imagen que se despliega en la pantalla y la envía hacia otro dispositivo para su despliegue, tomando esta idea y adicionando otras características para manejo de eventos, surge el protocolo VNC, que se limita a enviar la imagen con la resolución del cliente hacia la pared de vídeo. Algunos escritorios remotos como X DMX y Apple Remote Desktop emplean el protocolo VNC [Richardson et al., 1998] para su funcionamiento. El framework más representativo que emplea el mismo principio es SAGE ( Scalable Adaptative Graphics Environment ) [Leigh et al., 2008], este proyecto usa redes multigigabit [Smarr et al., 2003] para soportar el envío y recepción en alta resolución de las imágenes. Además emplea una alberca de memoria donde almacena todas las imágenes que se despliegan en la pared de vídeo, de modo que permite su movilidad sobre toda el área de despliegue.

Para el funcionamiento del Frame Buffer Compartido es necesario emplear estrategias para el envío de la imagen en la red. Dependiendo de la resolución del Frame Buffer de origen, el paquete puede ser demasiado grande, es por eso que algunos proyectos funcionan

adecuadamente solo con redes multigigabit, OptIPuter (Red óptica de velocidad multigigabit) es un proyecto basado en redes ópticas que permite el uso de una red de 10 Gb, lo cual implica una infraestructura especial para realizar el envío, transmisión y recepción del mensaje, así como una adecuada cantidad de memoria para recibir la información y almacenarla en un cache de alto desempeño, LambdaRam ( Alberca de memoria cache ) proporciona esa funcionalidad.

Con respecto a la GUI, la más conocida y popular es el escritorio estándar, ya que proporciona beneficios en el desempeño del usuario, además la mayoría de los usuarios están familiarizados con el. Por lo que tener un escritorio funcionando en la pared de vídeo puede facilitar su uso a la mayoría de los usuario, esto no se puede lograr directamente con el Render Paralelo, ya que no permite el funcionamiento de diversas aplicaciones en la misma área. La otra opción proporcionada por el Frame Buffer Compartido esta implementada en los escritorios remotos de X DMX [Faith and Martin, 2003] y Apple Remote Desktop, pero están limitados por el protocolo que usan VNC, ya que mantiene la resolución del cliente, la cual es mucho menor a la resolución total de la pared de vídeo, si no lo hicieran así los recursos necesarios en su funcionamiento se incrementarían demasiado como en el caso de SAGE Network 10 Gb, Cache 1 TB.

Una solución que puede funcionar es tener un escritorio distribuido, de manera que podamos mantener dos modos de funcionalidad combinados, como en un escritorio estándar (full screen, window server). De esta forma podríamos mantener la generación de la imagen para emplear el Render Paralelo o el FBS cuando sea necesario. Para lograrlo es necesario pensar en el manejador de ventanas distribuido, que es el encargado de la interacción entre todas las aplicaciones que se están ejecutando. Necesitamos un modelo que permita la creación de este manejador de ventanas distribuido, este modelo requiere diversos bloques independientes encargados de las funcionalidades básicas necesarias de comunicación, despliegue, distribución de datos, manejo de eventos y control (ver figura 13).

Existe un componente clave en el desarrollo de las interfaces gráficas, este componente permite el manejo de eventos y el despliegue, además es capaz de contener objetos del mismo tipo, por otro lado tenemos definiciones de objetos distribuidos que proporcionan la funcionalidad para la comunicación remota y la distribución de datos. De ambos conceptos podemos obtener la funcionalidad que buscamos, nosotros los llamamos objetos distribuidos visuales.

De lo cual podemos enunciar la siguiente hipótesis:

*Hipótesis: Podemos crear un modelo de objeto que sea utilizado como base en la creación de un manejador de ventanas distribuido para las paredes de vídeo.*

| Características | Render Paralelo | Memoria Compartida |
|---|---|---|
| Múltiples ventanas | No | Si |
| Distribución del render | Si | No |
| Requerimiento de transferencia de datos | Regular | Alto |
| Despliegue de cualquier tipo de información | No | Si |

Figura 13: Características del render paralelo y de la memoria compartida

# Objetivos de la investigación

## Objetivo general

Definir un modelo de objeto que sea la base para la creación de un manejador de ventanas distribuido, a este modelo le hemos denominado objeto distribuido visual (Distributed visual object, DVO).

## Objetivos específicos

Para el alcance del objetivo general, se requiere realizar los siguientes objetivos específicos:

- Analizar e identificar el componente básico empleado en el funcionamiento de los principales manejadores de ventanas para escritorios estándar.
- Analizar, identificar y obtener los requerimientos básicos para los manejadores de paredes de vídeo.
- Definir las funcionalidades básicas requeridas para el manejo de paredes de vídeo.
- Definir y formalizar los conceptos necesarios para describir el componente básico que permita realizar el manejo de paredes de vídeo.
- Diseñar una solución que permita, la distribución de los datos del cliente a los servidores, de manera que, la carga de trabajo sea distribuida en los servidores (render).
- Diseñar una solución para el manejo de eventos, que se deben comunicar del cliente hacia los servidores, manteniendo el uso del ancho de banda moderado.

- Definir el modelo de los objetos distribuidos visuales (DVO), contemplando los aspectos de distribución de datos, y manejo de eventos.
- Validar el nuevo modelo, de objetos distribuidos visuales, mediante una aplicación.
- Proporcionar las clases iniciales de la API de la capa DVO.

## Solución propuesta

Para solucionar las deficiencias en el desarrollo y utilización de las aplicaciones de paredes de vídeo, se propone el diseño e implementación de un modelo en capas como base para la creación de un manejador de ventanas distribuido. Las capas sugeridas en el modelo están divididas por funcionalidades especificas : *comunicación, despliegue (interfaz del objeto visual), contenedor, manejo de eventos y control.*

1) La capa de *comunicación* debe permitir la distribución de los datos, para repartir la carga de trabajo, proporcionando funcionalidades remotas basadas en diferentes protocolos de red (MPI, TCP/IP).

2) La capa de *despliegue* debe ser la encargada de la representación visual que ocurra del lado del cliente y debe haber una capa similar del lado de los servidores. Este modelo contempla como primitivas de objetos visuales a las imagenes, vídeos y modelos 3D, de tal forma que nos enfocaremos en el despliegue de este tipo de objetos.

3) La capa de *manejo de eventos* se encarga de recibir eventos almacenados en la cola de eventos generada por el sistema operativo, posteriormente serán respondidos y replicados a los servidores. La información obtenida en la cola de eventos, debe ser comparada con nuestra tabla de eventos para reconocer que tipo de evento ocurrió y realizar la acción adecuada para cada evento.

4) La capa *contenedor* es la encargada de almacenar los objetos visuales a desplegar, dentro de esta capa se guarda una relación de todos los objetos desplegados, de esta manera se pueden buscar los mismo objetos del lado de los servidores para replicar el comportamiento.

5) La capa de *control* es la encargada de relacionar las capas inferiores de comunicación y manejo de eventos con la capa de despliegue.

El conjunto de capas del cliente es similar al conjunto de capas del servidor exceptuando que la capa de *manejo de evento* solo funciona del lado del cliente. Además la capa *contenedor* en el cliente recibe el manejo de eventos directamente en los objetos, y en la capa *contenedor* de los servidores primero se debe localizar la replica del objeto, dentro

del arreglo de objetos desplegados y posteriormente ejecutar la acción generada por el evento.

Algunos patrones de diseño [1] pueden ser empleados en la construcción de estas capas. Por ejemplo *composite* nos permite manipular un compuesto de objetos como si fuera uno solo, de esta manera permitiremos el anidamiento de los objetos visuales. *Observer* permite que el control reconozca cuando un objeto visual del cliente recibe un evento. *State* puede emplearse para controlar el estado de los objetos de la capa de *comunicación*. *Strategy* se puede usar para identificar y trabajar con alguno de los objetos específicos que se mencionaron (imagen, vídeo, modelo 3D). En general podemos dividir nuestras capas en el patrón de diseño MVC (Model-View-Control), siendo la vista la capa de *despliegue*, la capa de *control* y las capas restantes son el modelo ya que son quienes proporcionan los datos del lado del servidor y de la GUI.

Estas capas permiten la creación de aplicaciones básicas para despliegue de los objetos visuales permitidos (imagen, vídeo, modelo 3D). El anidamiento y la jerarquización son características que permiten la extensión de estas capas a un manejador de ventanas distribuido.

Estas capas se mantienen dentro de cada componente del modelo cliente-servidor. Este modelo facilita el uso de la interfaz gráfica a los usuarios, manteniendo conceptos conocidos de un escritorio estándar.

Las lecturas encontradas al respecto reconocen que el escritorio estándar a probado mejorar el desempeño de los usuarios[Myers et al., 2000] [Myers, 1995] [Shneiderman, 2000], es por eso que es ampliamente usado, pero en el caso de la pared de vídeo algunas de sus características no son tan viables, es por eso que las funcionalidades básicas solo contemplan el uso de ventanas. Las ventanas controladas desde el servidor permitirán mantener el uso familiar de la GUI para el usuario.

La programación orientada a objetos ofrece una gama de propiedades que son necesarias para la implementación del modelo y de sus características. La herencia y la composición son básicas para obtener los objetos que estarán en la capa *contenedor y despliegue*. La separación de la interfaz y la implementación es fundamental en el manejo remoto de métodos empleado en la capa de *comunicación*. Dentro de la capa de *comunicación* se contempla la definición de un protocolo que permita el manejo de los métodos remotos entre los objetos del cliente y del servidor, este protocolo debe ser dirigido dependiendo de la petición realizada (envío de datos o respuesta a evento).

Finalmente, la propuesta se basa en el cumplimiento de características que pueden ser imitadas de comportamientos encontrados en objetos distribuidos y objetos visuales, por lo que al modelo se llamo *Objetos Distribuidos Visuales*.

# Contribuciones

La contribución es la creación de un nuevo modelo, este modelo se propone como base para la construcción de un manejador de ventanas distribuido, por lo que sienta las bases

[1] Son modelos que proporcionan soluciones a problemas comunes en el desarrollo del software.

para lograr un escritorio distribuido para las paredes de vídeo. Además por sus características implementadas en capas puede ser usado para facilitar la generación de aplicaciones básicas de despliegue.

El diseño del modelo del DVO esta pensado para extenderse a una API, habilita la posibilidad de la escalabilidad y deja abiertas las puertas para mejorar la interfaz de usuario (agregando nuevos dispositivos de entrada), ya que se maneja una tabla de valores que puede ser incrementada para el reconocimiento de nuevos eventos de usuario.

Fundamentalmente podemos resumir las contribuciones de la tesis en base al manejador de ventanas distribuido, y a sus aplicaciones, como se ve a continuación:

- *Un modelo DVO como base para la creación del manejador de ventanas distribuido.*

  DVO define un modelo, que reduce algunas limitaciones de las demás aplicaciones, que funcionan sobre una pared de vídeo, al permitir el manejo de diferentes aplicaciones al mismo tiempo, conservando alta resolución.

- *Implementación de una aplicación empleando el modelo DVO*

  Se implementó un visor de imagenes en objetive-C, que permite el despliegue de imagenes de alta resolución sobre la pared de vídeo en la plataforma MAC OSX.

- *Formalización de los conceptos en base al paradigma orientado a objetos*

  En base a esta formalización es posible validar la disminución en el tamaño del mensaje entre el cliente y el servidor utilizando el modelo DVO.

- *Bases para una API DVO*

  Se proponen las bases necesarias para la creación del API DVO.

- *Algoritmo de distribución de datos*

  Se diseñó un algoritmo para la distribución de los datos tomando en cuenta la posición de las pantallas en la pared de vídeo.

## Organización de la tesis

A continuación se muestra la organización del documento, dividido en capítulos que explicarán el desarrollo de esta investigación.

- Introducción.- En esta parte se muestran detalles sobre la investigación, el planteamiento del problema, la solución propuesta, contribuciones y objetivos. Así como una breve introducción a la tecnología de manejo de pared de vídeo existente en la actualidad.

- Capítulo 1 (Estado del Arte).- En este capítulo se explican detalles sobre las principales vertientes existentes para el manejo de las paredes de vídeo, se hace un recuento de características desarrolladas hasta ahora por algunos proyectos, así como una comparación entre ellas.

- Capítulo 2 (Entorno de desarrollo del modelo DVO).- En este capítulo se resumen temas específicos de visualización, interacción humano-computadora, sistemas distribuidos, patrones de diseño y redes. Con los que se trabaja durante el desarrollo del modelo.

- Capítulo 3 (Análisis).- En este capítulo se hace el análisis de los diferentes proyectos encontrados en el estado del arte, haciendo un enfoque en los bloques funcionales que representan la arquitectura de un manejador de pared de vídeo. También se presenta el funcionamiento de un prototipo de visor de imágenes que representa una funcionalidad de la pared de vídeo.

- Capítulo 4(Modelo).- En este capítulo se describe el modelo propuesto por medio de diagramas que permiten explicar su funcionamiento en cada capa, así como una formalización de los conceptos involucrados en el modelo.

- Capítulo 5(Visor de imágenes).- Se hace la caracterización de una aplicación básica en las paredes de video. Se explica el uso de algunos patrones de diseño implementados.

- Capítulo 6(Conclusiones).- Se listan los resultados y las conclusiones obtenidas en el trabajo, así como la discusión y en trabajo a futuro.

Capítulo 1

# Estado del Arte

## 1.1. Funcionamiento de la pared de vídeo

Existen ciertos tipos de visualización que requieren alta resolución y grandes dimensiones para su despliegue, por ejemplo las imágenes obtenidas vía satélite. Este tipo de imágenes requieren su despliegue sin redimensionamiento de la imagen original para evitar perdida de información. De ahí surge la importancia de herramientas que cumplan con esas características de despliegue, una herramienta que cumple con esos requisitos es la pared de vídeo [Czerwinski et al., 2005].

La pared de vídeo como herramienta de visualización proporciona múltiples beneficios en la productividad del usuario [Czerwinski et al., 2003], [Kukimoto Nobuyuki, 2010], pero aún no existen estándares que definan como debe ser su interacción con el usuario. Algunas investigaciones tratan de mejorar el diseño de esta interacción tomando en cuenta la escalabilidad del despliegue [Sakuraba et al., 2011], el contexto en el que se encuentra el usuario [Wong et al., 2007] y los flujos de trabajo involucrados en el proceso de obtención y despliegue de la información [Nguyen et al., 2011]. Hasta el momento estos trabajos no definen como deben ser los procesos de interacción con el usuario, si no que realizan pruebas tomando en cuenta sus propias perspectivas de diseño.

La pared de vídeo se ha utilizado en múltiples experimentos cuantitativos y cualitativos, para tratar de obtener evidencia que demuestre la relación que existe entre el cambio en los efectos visuales y la productividad de los usuarios en trabajo colaborativo e individual [Ni et al., 2006], [Yuill and Rogers, 2012], [Huang et al., 2006], [Huang, 2005]. Otro trabajo que experimento con las paredes de vídeo para probar que esta herramienta incrementa la percepción del usuario fue mostrado en [Baudisch et al., 2003].

La pared de vídeo ha evolucionado conforme a sus componentes principales (las pantallas), ya que la resolución de las pantallas se ha ido incrementando gradualmente. Las pantallas que son utilizadas comúnmente en las paredes de vídeo son LCD o LED de alta resolución, aunque es posible mantener un arreglo de proyectores para el despliegue[1].

[1]El problema de los proyectores es principalmente el costo y el espacio requerido para el despliegue

La mayoría de las aplicaciones que se utilizan para desplegar objetos en tres dimensiones utilizan la biblioteca de gráficos OpenGL. Aunque existen otras bibliotecas que pueden ser utilizadas como Open Scene Graph (OSG)[Wang and Qian, 2010]. Otros enfoques de funcionamiento de paredes de vídeo mantiene una solución basada en el hardware como Teravision[Singh et al., 2003]. Así como proyectos que proponen estrategias de distribución y control de datos como XmegaWall [Kang and Chae, 2007].

En conclusión los diversos trabajos encontrados presentan información que muestra los beneficios que proporciona el uso de la pared de vídeo. Además se muestran los caminos que se han seguido para tratar de encontrar definiciones que describan la interacción entre la pared de vídeo y el usuario, pero hasta ahora no hay un estándar que lo defina.

## 1.2. Resolución de la pared de vídeo

La pared de vídeo proporciona una alta resolución uniendo las resoluciones del número de pantallas que la forman, de modo que es posible aumentar la resolución de la pared, aumentando el número de pantallas o aumentando la resolución de cada pantalla. Cuando se incrementa el número de pantallas, se debe incrementar también el número de nodos que las controlan. Cuando la resolución de cada pantalla aumenta, se requiere aumentar la capacidad de la tarjeta gráfica del nodo de control.

OptIPrecence es la pared de vídeo de la universidad de San Diego California (Calit2), fué la pared de vídeo más grande hasta el 2010 (ver figura 1.2) con una resolución de 315,648,000 pixeles y 82 pantallas con una resolución de 2,560x1,600 cada una. Trabaja sobre la plataforma linux y emplea el middleware de CGLX, el cual es utilizado en un amplio conjunto de paredes de vídeo.

Por ejemplo en el caso del CinvesWall tenemos que cada nodo del cluster de visualización tiene una resolución de 1920 por 1200 lo que da una resolución por pantalla de 2,304,000 equivalente a 2 Megapixeles que multiplicada por la profundidad de color resulta en la capacidad de memoria del frame buffer por 32 bits de profundidad da un total de 73,728,000 equivalente a 73 Megabits para cada despliegue.

Para la pared de vídeo completa tenemos la unión de la resolución de las 12 pantallas es decir, 2,304,000 resolución por pantalla por 12 que es el número de pantallas de la pared de vídeo, da un total de 27,648,000 equivalente a 27 Megapixeles con un almacenamiento global de frame buffer de 884 Mb para la pared de vídeo completa.

Un estimado en el crecimiento a futuro de las pantallas de alta resolución es proporcionado por Intel, mostrando que para el 2015 se llegará casi a 12 Mega Pixeles en pantallas de 30 pulgadas (figura 1.2).

En conclusión las paredes de vídeo incrementan su resolución acumulando la resolución de cada pantalla, existen muchas paredes de vídeo que están siendo utilizadas por múltiples

| Nombre | Resolucion (pixels) | Número de pantallas | Resolución por pantalla | Tecnología | Universidad |
|---|---|---|---|---|---|
| OptIPresence | 315,648, 000 | 82 | 2,560 x 1,600 | Linux/CGLX | Calit2, UC San Diego |
| Stallion | 307,200, 000 | 75 | 2,560 x 1,600 | Chromium/ Paraview/ SAGE | TACC, University of Texas |
| HiperSpace | 286, 720, 000 | 70 | 2,560 x 1,600 | Rocks/Linux CGLX | Calit2 UC San Diego |
| Hyperwall-2 | 256,000, 000 | 128 | 1680x1050 | SUSE Linux/ NFS/DHCP | NASA Ames |
| HyperWall | 64,225, 280 | 50 | 1280x1024 | Chromium | Calit2 UC Irvine |
| CinvesWall | 27,648, 000 | 12 | 1920x1200 | MacOSX/ DVO | |

Figura 1.1: Resoluciones en paredes de vídeo hasta el 2010

| Market Segment | Panel size | 2011 | 2012 | 2013 | 2014 | 2015 |
|---|---|---|---|---|---|---|
| Premium | 24-30" | 2560x1440 | 2560x1440 | 3840x2160 | 4096x2304 | 4800x2700 |
| MS | 21-24" | 1920x1080 | 1920x1080 | 2560x1440 | 2560x1440 | 3840x2160 |
| Entry | 17-21" | 1366x768 | 1366x768 | 1366x768 | 1920x1080 | 2560x1600 |

Figura 1.2: Resoluciones de pantalla estimadas hasta el 2015 por Intel

universidades.

## 1.3. Tipos de manejo de paredes de vídeo

### 1.3.1. Basado en render paralelo

Render, es el nombre que se le da al proceso de convertir un modelo 3D, en una matriz bidimensional de pixeles, que puede ser desplegada en la pantalla. Existen diferentes frameworks dedicados al render multiescala [Hsu et al., 2011], que emplean diversas técnicas de visualización [Lipsa et al., 2011] como data streaming [Ahrens et al., 2001],

[Callaghan et al., 2002], [Beringer and Hullermeier, 2006], [Nouanesengsy et al., 2011], enfocados en diversas aplicaciones, por ejemplo la exploración [Fu and Hanson, 2007].

El render requiere un gran número de operaciones, ya que un objeto 3D, generalmente tiene textura, iluminación, etc. De forma que estas transformaciones requieren múltiples operaciones matriciales, por lo que al aumentar el tamaño del modelo, incrementa el número de operaciones a realizar. Así que, para lograr desplegar y manipular un modelo en 3D, dentro de una pared de vídeo de grandes dimensiones, se requiere de grandes cantidades de procesamiento.

Para poder emplear un render paralelo debemos trabajar sobre un cluster de visualización, ya que permite realizar la generación de la imagen de forma paralela. Cuando se trabaja el render paralelo, no necesariamente los datos de entrada, son procesados completamente en el cluster de visualización, de modo que, es posible recibir los datos con un pre-procesamiento desde otros dispositivos.

Monlar et al [Molnar et al., 1994] definen una taxonomía de procesos para el render paralelo, esta taxonomía resume los diferentes procesos que se deben realizar durante el render, divide el render paralelo en 3 tipos de algoritmos (figura 1.3). El primero es el algoritmo Sort-first usado por WireGL y Chromium, con este algoritmo la distribución del render paralelo, se realiza desde la base de datos gráfica, es decir la distribución ocurre al inicio del proceso de render. El segundo es el algoritmo Sort-middle, este algoritmo obtiene la información a distribuir con un pre-procesamiento, de modo que las transformaciones geométricas ya fueron realizadas. El tercero es el algoritmo Sort-last, este algoritmo obtiene la matriz de despliegue y la distribuye entre los nodos de despliegue, este ultimo proceso se conoce como rasterización.

- Sort-first: Es un algoritmo de render paralelo, que funciona mediante el envio de los datos a desplegar antes de la rasterización. De esta forma lo que se distribuye en este algoritmo es el conjunto de primitivas básicas de dibujo, almacenadas en la base de datos gráfica de origen. En este punto es posible separar los vértices y líneas que forman el modelo, para enviarlos a cada uno de los nodos, una de sus implementaciones se conoce como pipeserver[Mueller, 1995].

- Sort-middle: Es un algoritmo de render paralelo, que funciona mediante el envio de datos durante la rasterización, es decir se realizan las transformaciones geométricas y los resultados son divididos y enviados a los nodos. De esta manera lo que se envían son formas geométricas listas para ser agrupadas.

- Sort-last: Es un algoritmo de render paralelo, que funciona mediante el envio de datos después de la rasterización, en este caso ya se tiene la matriz de pixeles, la cual se fragmenta para ser enviada a los nodos. Un ejemplo de implementación es la biblioteca libpglc [Wylie et al., 2001]. A diferencia del enfoque basado en Frame buffer shared, lo que se envía es solo una fracción de la matriz de pixeles original. Cada fracción representa la resolución por pantalla (que forma la pared de vídeo), también depende de la configuración matricial de la pared de vídeo.

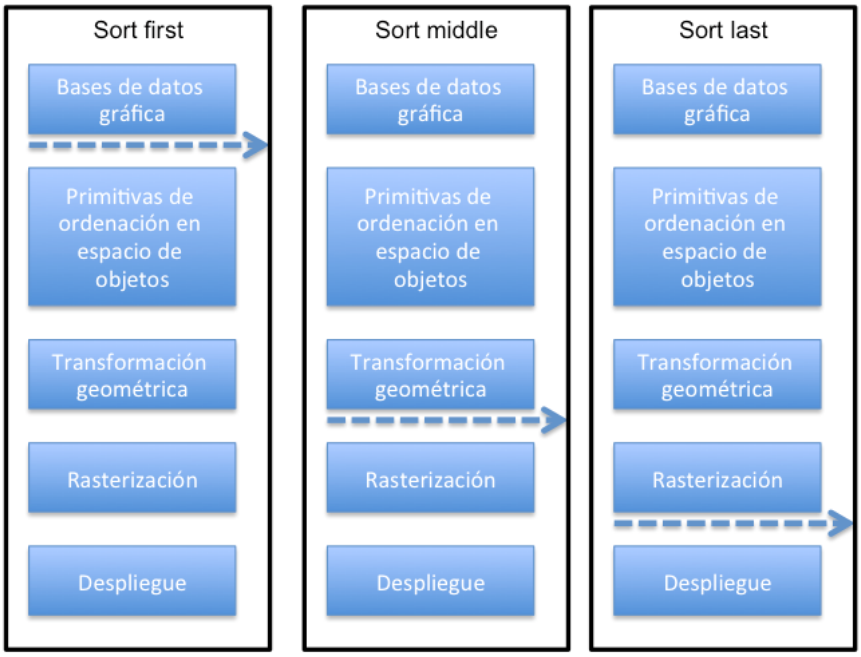


Figura 1.3: Taxonomía de algoritmos para render parallelo Sort-first, Sort-middle, Sort-last

Se debe evaluar el uso de cada algoritmo dependiendo de las capacidades de los componentes. Por ejemplo, si tenemos un servidor de vídeo de mayor capacidad que los nodos hablando de los nodos individualmente, la mejor opción es usar un algoritmo Sort last. Cuando cada nodo tenga mayor capacidad que el servidor de visualizaciones es mejor emplear un algoritmo Sort first, ya que de ese modo se delega el trabajo de Render en los nodos.

WireGL [Humphreys et al., 2001] [Humphreys et al., 2000] Stanford University Computer, Graphics Lab, es una biblioteca escalable para render paralelo, proporciona un ambiente similar al del OpenGL. Realiza la virtualización de múltiples aceleradores gráficos usando el algoritmo de render paralelo sort-first, con una interfaz paralela. Esta biblioteca es una extensión de OpenGL y es una de las primeras herramientas basadas en el render paralelo. Funciona mediante servidores gráficos, llamados pipeservers que trabajan sobre múltiples protocolos como TCP/IP y Myrinet.

Chromium [Humphreys et al., 2002] Lawrence Livermore National Laboratory, Stanford University, University of Virginia, Tungsten Graphics, Inc. es una API para manipulación de flujos de gráficos para cluster de visualización, para algoritmos sort-first y sort-last.

JuxtaView [Krishnaprasad et al., 2004] Electronic Visualization Laboratory University of Illinois at Chicago, Earth Resource Observation Systems, Data Center US Geological Survey, es una aplicación paralela basada en cluster para visualizar imágenes en alta

resolución, emplea cómputo paralelo y memoria distribuida. Su software esta basado en niveles de red de la memoria cache, está construida sobre las bibliotecas de MPI, OpenGL, GLUT y SDL. Los comandos son enviados vía broadcast a todos los procesos esclavos.

MPK [Bhaniramka et al., 2005] es un conjunto de herramientas para render paralelo escalable basados en OpenGL. MPK proporciona una API para manejar aplicaciones gráficas mediante diferentes subsistemas gráficos. MPK soporta diferentes modos de particionamiento y algoritmos de composición optimizados basados en GPU.

Equalizer [Eilemann et al., 2008] Eyescale Software and Visualization and MultiMedia Laboratory (VMML), University of Zurich, es un framework que permite el render paralelo basado en OpenGL, proporciona una API para desarrollar aplicaciones para las paredes de vídeo, sobre clusters de visualización distribuida y multiprocesadores paralelos. Este framework se diseño lo más genérico posible para permitir el desarrollo de aplicaciones de render paralelo con diferentes tipos de datos.

Power Wall [Lawlor et al., 2008] University of Minnesota, Silicon Graphics, Inc., IBM Storage Products Division, usa MPI-GLUT que es la unión de OpenGL 3D GLUT y la comunicación MPI, MPIglut envía los eventos entrantes sobre un socket TCP a los procesos internos, donde los eventos son replicados en broadcast vía MPI a los nodos de despliegue. MPIglut se construye sobre la implementación secuencial de GLUT. La mayoría de los eventos MPIglut son entregados colectivamente al usuario en coordenadas globales.

CGLX [Doerr and Kuester, 2011] Center of Graphics, visualization and Virtual Reality (GRAVITY), California Institute for Telecommunications and Information Technology, University of California, San Diego. Emplea OpenGL-Utility-Toolkit ( GLUT) para desplegar aplicaciones de render distribuido. Proporciona una herramienta de configuración para facilitar el uso del Grid. Permite la escalabilidad y desempeño interactivo. El diseño independiente de la plataforma y el hardware. Permite soporte de sistemas heterogéneos. Emplea el modelo maestro-esclavo. CGLX intercepta llamadas OpenGL para generar y manejar el contexto OpenGL en el sistema de visualización.

Algunas de las desventajas de la mayoría de las aproximaciones basadas en el render paralelo son las siguientes:

- Soporte de aplicaciones genéricas.- Las paredes de vídeo son herramientas de visualización que suelen utilizarse en diversas áreas de investigación, cada área proporciona problemas específicos con diversas estructuras de datos. Esta variedad de problemas ha provocado que la mayoría de las soluciones funcionales en las paredes de vídeo se enfoquen en un dominio de solución específico, por esta razón se esta buscando una aproximación más general.

- Abstracción escalable de la capa de los gráficos.- Este es uno de los puntos más importantes del render paralelo, ya que en la mayoría de los casos los gráficos involucran toda el área de despliegue y no es posible modificarlos. De modo que la capa de los gráficos es un poco estática en ese aspecto.

- Explotar infraestructuras de código existente, tal como estructuras de datos mole-

culares, nivel de detalle (LOD) y bases de datos geométricas.- La variedad de datos de entrada que se requieren desplegar es muy amplia, y en algunos casos se debe tomar ventaja de las estructuras originales.

Muchas investigaciones desarrolladas sobre paredes de vídeo han empleado técnicas de render paralelo. Por ejemplo WireGL o Chromium, entre otros, durante su ejecución se requiere la creación de un ambiente de comunicación remota. Algunos experimentos han verificado resultados y deficiencias sobre todo en la reproducción de vídeo [Ebara, 2011].

Resumiendo, el render paralelo divide el proceso de generación de la imagen en el número de nodos que se tengan para el despliegue en la pared de vídeo. La biblioteca empleada usualmente en el render paralelo es OpenGL. Este tipo de despliegue suele ser estático en cuanto a la posición del despliegue final. Cuando la visualización es de un modelo en 3 dimensiones y se despliega sobre toda el área de la pared de vídeo este enfoque es la mejor opción.

### 1.3.2. Basado en memoria compartida

El frame buffer es parte de la memoria de la tarjeta gráfica, contiene la matriz de pixeles que será desplegada en pantalla. El tamaño del frame buffer de una tarjeta gráfica, se obtiene mediate la multiplicación de los pixeles horizontales, verticales y la profundidad, en algunos casos se ha trabajado con procesadores RISC con el fin de generar *smart frame buffer* que son capaces de expandir su tamaño[McCormack and McNamara, 1993]. Cuando una visualización se despliega sobre pantalla, se requiere de cierto procesamiento para generar la matriz de pixeles que se mostrarán, los GPUs son los encargados de este procesamiento. La arquitectura de las tarjetas NVidia por ejemplo (figura 1.4), consta de dos bloques principales, la memoria de vídeo y el cache de memoria.

Cuando el despliegue de visualizaciones de alta resolución y grandes dimensiones se hace de manera remota, como en el caso de la pared de vídeo, es necesario emplear un ancho de banda que en algunos casos supera los 10GB. Esta cantidad es obtenida por redes ópticas como Optiputer [Smarr et al., 2003]. Los estudios de lightpaths [Naiksatam et al., 2005] y lightspeed [Papadopoulos et al., 2004] son investigaciones realizadas para el reconocimiento de la longitud de onda (lambdas) empleado en redes ópticas, por lo que el uso de este tipo de redes requiere infraestructura especializada para multiplexar la señal de salida y de entrada de la red óptica, para ser desplegada en múltiples pantallas (OptIPortals)[DeFanti et al., 2009]. Otras investigaciones han tratado de mejorar el ancho de banda implementando paso de mensajes sobre redes genéricas Ethernet como Open-MX [Goglin, 2008][Goglin, 2009].

Otra necesidad generada por el despliegue remoto de gran tamaño es la de la memoria, ya que se requieren grandes cantidades de memoria para almacenar el flujo de información entrante. LambdaRAM[Vishwanath, 2009] es un proyecto que proporciona memoria cache distribuida de alto desempeño para los nodos de uno o más clusters conectados mediante redes de alta velocidad, permitiendo que las aplicaciones puedan acceder a los datos de la memoria local y remota.

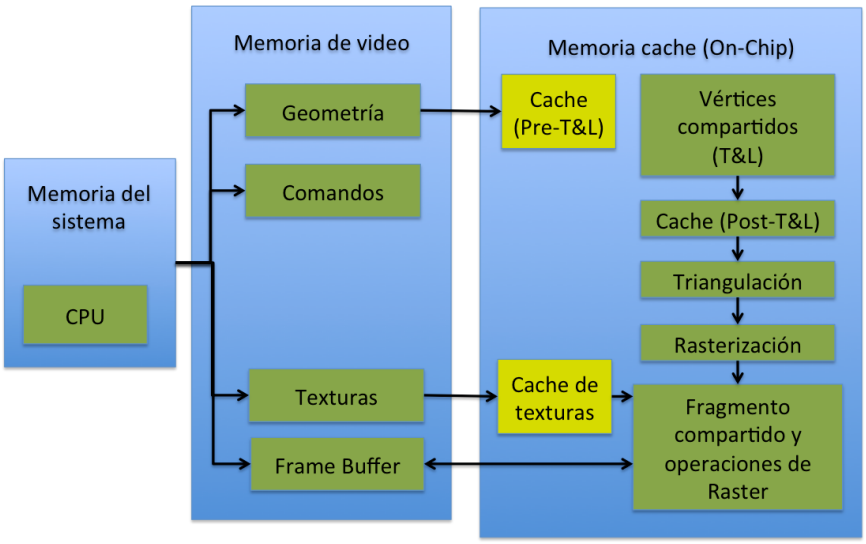


Figura 1.4: Arquitectura del GPU (NVidia)

El protocolo básico que se emplea para el acceso remoto a interfaces de usuario gráficas es RFB (Remote Frame Buffer)[Richardson, 2010]. Este protocolo es adoptado por el protocolo VNC[Richardson et al., 1998], los escritorios remotos de Apple ® y Microsoft ® emplean VNC principalmente para el manejo remoto de eventos. Otros escritorios remotos como XDMX[Faith and Martin, 2003] definen un servidor proxy que permite a los servidores X (X windows system) combinarse dentro de un servidor X multi-headed [Gettys et al., 1990], que en combinación con Xinerama dan la apariencia de tener una gran pantalla de alta resolución.

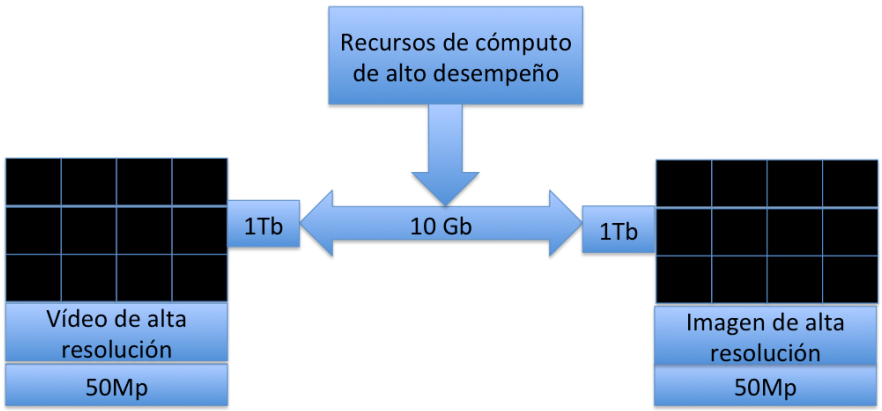


Figura 1.5: Estructura de despliegue empleado redes de alta velocidad

Una arquitectura de visualización ultraescala colaborativa [Jeong et al., 2010] (figura 1.5), consta de paredes de vídeo empleadas para el despliegue y recursos de cómputo de alto desempeño encargados de la generación de la visualización (High-performance-computing resource). SAGE es un proyecto que asume que, si el costo de la red es menor al costo de cómputo y almacenamiento, es más económico compartir la información por medio de la red que instalar y mantener recursos de cómputo locales.

SAGE [Leigh et al., 2008] (2008) : (Scalable Adaptative Graphics Environment ) SAGE es un desarrollo del Electronic Visualization Laboratory de la University of Illinois at Chicago, es un ambiente de ventanas que permite desplegar diversas aplicaciones al mismo tiempo, basado en el uso de redes multigigabit. SAGE usa frame buffer sharing (figura 1.6), esta formado por diferentes sistemas encargados del despliegue y distribución de los datos, en este caso pondremos atención en el módulo de FreeSpaceManager, este módulo trabaja como una alberca de memoria que almacena el contenido del frame buffer de los dispositivos donde se encuentre la interfaz gráfica del cliente. La información que se desplegará en la pared de vídeo es la que contiene la memoria del FreeSpaceManager, el flujo de pixeles es enviado hacia los encargados de la distribución del despliegue (SAGE Display Manager). La funcionalidad de SAGE puede ser descrita como pixel-streaming (flujo de pixeles), lo cual le da la ventaja de poder desplegar cualquier tipo de aplicación sin modificación alguna.

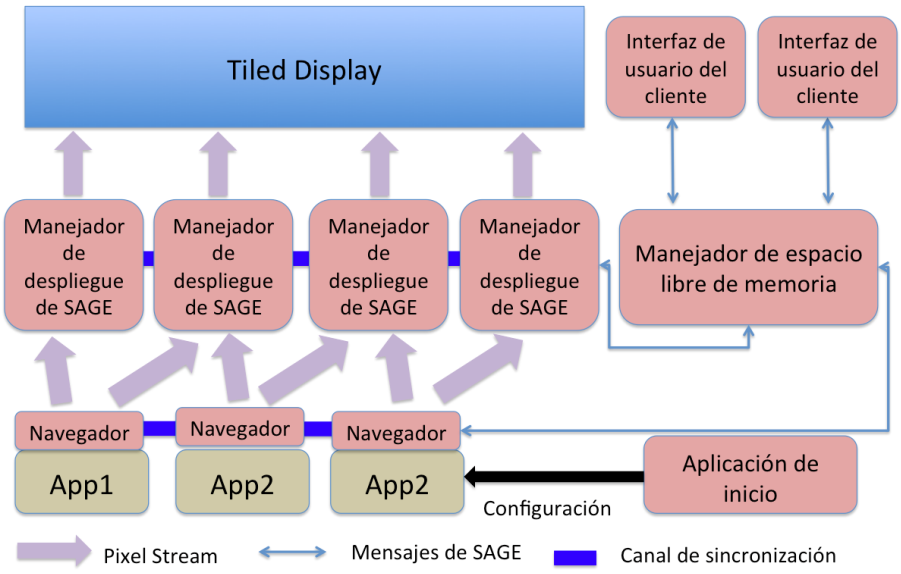


Figura 1.6: Arquitectura de Scalable Adaptative Graphic Environment (SAGE)

De modo que podemos definir el frame buffer sharing como el despliegue de la salida gráfica de algún dispositivo. Este enfoque permite tomar la matriz de pixeles lista para visualizar, de este modo cualquier tipo de visualización puede ser desplegada. La generación del gráfico depende únicamente del dispositivo que nos proporcione la salida. Uno de

los inconvenientes más marcados en este tipo de tecnologías es el uso indiscriminado de la red, ya que cuando el despliegue es en alta resolución el tamaño del paquete es grande. El tamaño del paquete esta definido por la resolución; la resolución es el número de pixeles que se despliegan en pantalla, cada pixel tiene una profundidad de 32 o 64 bytes. Por lo que el tamaño del paquete esta dado como el producto ( alto por ancho por profundidad ) de cada imagen, es por eso que se requiere el uso de redes ópticas multigigabits. Además el procesamiento implicado en la visualización no aprovecha las ventajas del cluster que maneja la pared de vídeo.

### 1.3.3. Basado en tarjetas gráficas en cascada

Cuando se requiere el despliegue de modelos en 3D, existen una variedad de técnicas para hacer el render. Específicamente en el render de animaciones con efectos visuales de algunas películas animadas, por ejemplo Monsters vs Aliens uso un total de 40 millones de horas en su proceso de render. De ahí la necesidad de mantener granjas de render. De hecho existen empresas que se encargan de hacer el render como Render Farm, para eso emplean 42 procesadores intel Xeon, y 12 terabytes de almacenamiento.

Este tipo de necesidades se solucionó usando directamente el GPU, que es la unidad de procesamiento gráfico que se encuentra dentro de la tarjeta gráfica. La idea es que todo el render se realice directamente dentro de los GPU's, por ejemplo trazo de rayo, es una técnica de render que produce imágenes con reflexiones y refracciones de luz perfectas, así como sombras y luces fotorealistas (figura 1.7)[2].

Figura 1.7: Render usando trazo de rayo

Pero el tiempo para su procesamiento hasta hace poco era demasiado, con la nueva arquitectura de Fermi de NVIDIA[Wittenbrink et al., 2011], la GeForce GTX 400/500

[2]http://www.nvidia.com/object/GTX400architecture.html

GPU pudo realizar trazo de rayo en tiempo real y obtener hasta 15 frames por segundo. Para un alto realismo visual, el GPU puede mantener una iluminación global. De ese modo las imágenes redefinidas son indistinguibles entre imágenes generadas por computadora y generadas por una cámara de vídeo.

Es posible unir dos tarjetas GeForce mediante SLI, SLI tiene un protocolo de comunicación inteligente dentro del GPU y alta velocidad en la interfaz digital para facilitar el flujo de datos entre dos tarjetas gráficas. Las tarjetas Quadro permiten ampliar el espacio de visualización mediante la tecnología NVIDIA Quadro Mosaic. Cualquier aplicación puede desplegarse a lo largo de 16 pantallas de alta resolución, desde una misma estación de trabajo y sin perder rendimiento ni calidad de imagen.

Se puede hacer uso de tarjetas en cascada con el QVM6800 que permite la configuración de 6 tarjetas Quad, para producir la salida de hasta 24 pantallas, de modo que la *escalabilidad* en resolución es limitada por la resolución de la tarjeta.

Existen también vídeo splitter (fraccionadores de señales de vídeo), son amplificadores de señales de vídeo que distribuyen la señal analógica de vídeo de una tarjeta gráfica a dos, cuatro u ocho monitores VGA y SVGA. Con la tecnología de tarjetas en cascada se pueden usar varias pantallas, pero como el trabajo estará centralizado, el desempeño disminuirá proporcionalmente al número de pantallas conectadas.

Una de las principales dificultades para depender únicamente de las tarjetas gráficas es que su costo es elevado, por ejemplo la nueva GeForce GTX 690 tiene un precio de 999 dólares, cada tarjeta.

## 1.4. Discusión sobre el manejo de paredes de vídeo

Los proyectos mencionados se han enfocado en lograr alguna de las características deseadas para el uso de las paredes de vídeo ( facilidad en el despliegue de cualquier tipo de aplicación, control transparente de toda el área de despliegue, uso moderado del ancho de banda, etc). Ninguno de ellos ha podido cumplir con todas las características ya que algunas de ellas en ocasiones son contrarias.

Una ventaja obtenida por un enfoque centralizado es básicamente la facilidad de despliegue de cualquier tipo de aplicación, esto gracias a que trabaja empleando una GUI extensamente conocida (escritorio estándar), de modo que existen múltiples bibliotecas y frameworks que trabajan sobre él. Hay opiniones encontradas en cuanto a la extensión del escritorio en ambientes de pared de vídeo, pero se ha demostrado que el manejo de ventanas proporciona beneficios en el desempeño del usuario, como se mencionó en secciones anteriores.

La principal problemática del enfoque centralizado es la escalabilidad (tamaño y resolución), este problema ha sido estudiado durante mucho tiempo [ ha dado origen a soluciones distribuidas]. La distribución en diversos aspectos (procesamiento, despliegue, resolución, etc.) se han convertido en un reto, ya que implica diversos procesos (sincronización, distribución, mensajes de control, etc). La mayoría de las bibliotecas, frameworks y middlewares basados en render paralelo tienen un enfoque distribuido, que por sus ca-

racterísticas dificulta obtener la facilidad en el despliegue de cualquier tipo de aplicación. Los recursos que emplean frame buffer shared son dependientes del ancho de banda de los mensajes, permiten el despliegue de diversas aplicaciones pero dependen totalmente de la fiabilidad de la red.

Depender de la red limita la arquitectura a un grupo limitado de usuarios, es por eso que se necesita encontrar una combinación para obtener las ventajas de ambos enfoques, a fin de obtener un modelo que permita la construcción de un *escritorio distribuido*. El escritorio distribuido puede manejar cualquier aplicación como en un escritorio estándar, y realizar de manera interna las operaciones necesarias para realizar la distribución. De este modo se pueden obtener las características deseables en una pared de vídeo.

Para poder obtener estas características en los siguientes capítulos trabajaremos sobre los siguientes temas:

- Manejo de eventos.
- Distribución de datos.
- Sincronización del sistema.
- Manejador de despliegue.

Así que el diseño de nuestro modelo tiene como finalidad proponer estrategias para el funcionamiento de la distribución de los datos, el manejo de eventos, sincronización y despliegue. En la (figura 1.8) se muestra un cuadro comparativo entre algunas soluciones existentes para el uso de las paredes de vídeo.

| Proyecto | Año | Tecnologías | |
|---|---|---|---|
| WireGL | 2001 | OpenGL | Biblioteca |
| Chromium | 2002 | Stream Buffer, OpenGL | Framework |
| Teravision | 2003 | Video Stream Buffer | Hardware |
| X DMX | 2003 | FBS (VNC) | Remote desktop |
| JuxtaView | 2004 | MPI, OpenGL, GLUT SDL | Aplicación |
| MPK | 2005 | OpenGL, OpenCL | Toolkit |
| XMegaWall | 2007 | Stream distribution, control data distribution | Biblioteca |
| SAGE | 2008 | FBS, pixel stream, LambdaRAM, Optiputer | Framework |
| Powerwall | 2008 | MPI, GLUT, TCP, broadcast | Biblioteca |
| Optiportal | 2009 | Stream Buffer, LambdaRAM, Optiputer | Aplicación |
| Garuda | 2010 | Open Scene Graph, multicast | Framework |
| CGLX | 2011 | GLUT extendido, multicast, UDP | Middleware |
| DVO | | MPI, TCP, UDP, OpenGL, Quartz | Middleware |

Figura 1.8: Comparativa de soluciones

El cuadro comparativo muestra los nombres de los proyectos utilizados comúnmente en el manejo de paredes de vídeo, además se muestran los años en los que se presentaron y el tipo de tecnología que utilizan. Se muestran también los paquetes en lo que están contenidos que van desde aplicaciones, bibliotecas, frameworks, middlewares y escritorios remotos. Al final del cuadro se incluye nuestra propuesta con el conjunto de tecnologías que se pueden utilizar.

Capítulo 2

# Entorno de desarrollo del modelo DVO

## 2.1. Entorno del modelo DVO

El modelo DVO se debe discutir en medio de un entorno de soluciones que se entretejen constituyendo toda una solución global a la propuesta de esta tesis. Es por eso que se deben discutir algunos temas relacionados con las áreas de visualización (junto con Computer Graphics), sistemas distribuidos (junto con la programación orientada a objetos), redes e interacciones humano-computadora (ver figura 2.1).

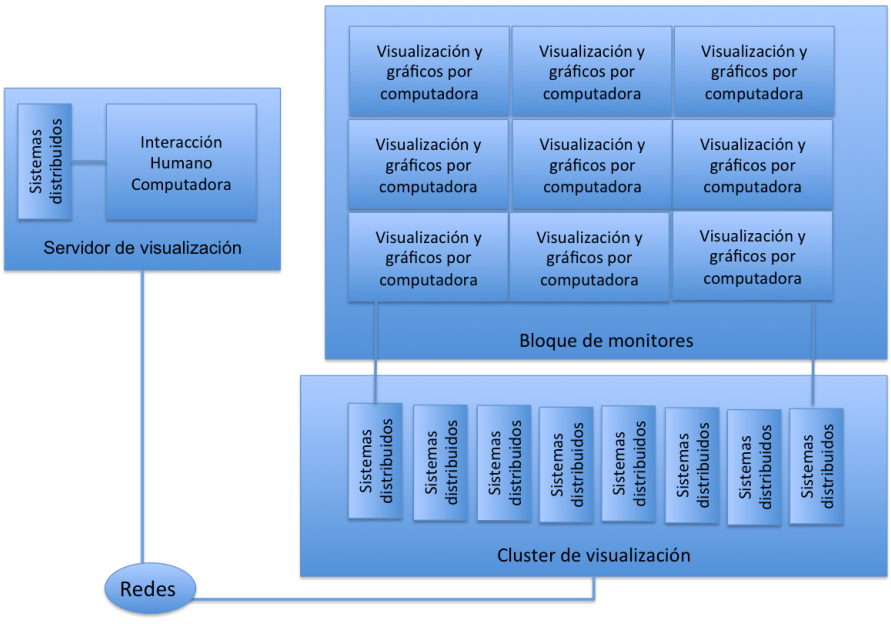


Figura 2.1: Entorno de desarrollo del modelo de objetos distribuidos visuales

La visualización y la graficación por computadora son las áreas de la computación que proporcionan los métodos y técnicas que permiten la transformación de datos en visua-

lizaciones desplegables. Las interacciones humano-computadora son la base que permite la comunicación entre el usuario y el sistema, esta interacción ocurre mediante la interfaz gráfica de usuario (GUI); en el caso de las paredes de vídeo es un tema abierto de investigación, ya que todavía no existe un estándar de GUI (por el contrario en máquinas mono usuario tenemos el escritorio estándar). Los sistemas distribuidos son los encargados de proporcionar las herramientas necesarias en la comunicación, esto mediante los mensajes remotos enviados entre los objetos cliente y servidor; los objetos distribuidos permiten el control de métodos remotos del cluster de visualización desde el servidor de visualizaciones. Las redes son las encargadas de proporcionar los protocolos empleados en la capa de transporte, con la finalidad de enviar los datos y los mensajes necesarios en el manejo de la pared de vídeo.

## 2.2. Temas principales para la definición del modelo DVO

### 2.2.1. Visualización y Graficación por computadora

A continuación se explicarán conceptos relacionados a la visualización en general, en la figura 2.2 se señala la parte principal de la arquitectura que requiere de estos conceptos.

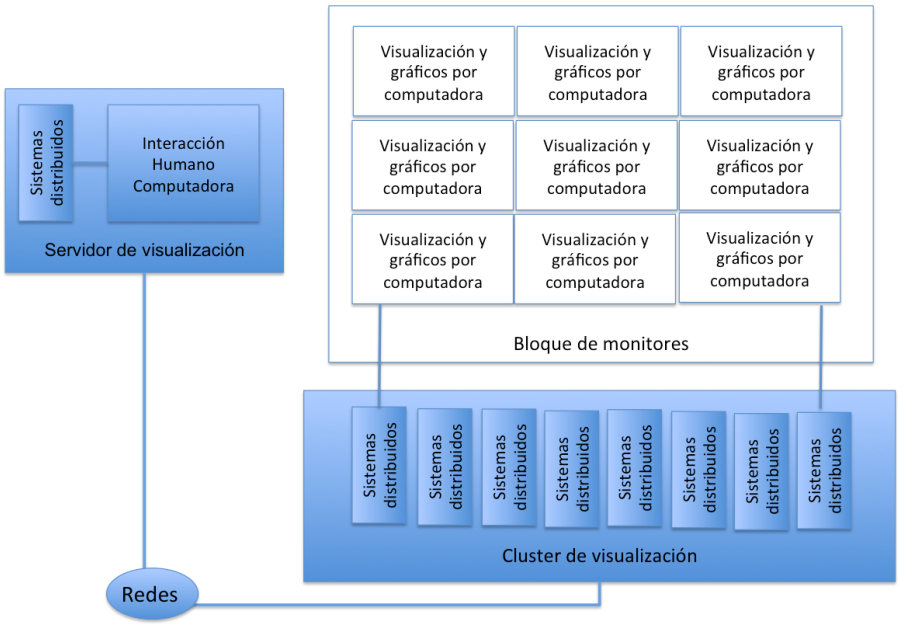


Figura 2.2: Conceptos relacionados a la Visualización y Graficación por computadora

La visualización en la ciencia se encarga de generar gráficas y modelos que permiten la explicación de observaciones, generación de previsiones y el entendimiento de teorías,

para ello se emplean diversas técnicas que permiten la representación de datos multidimensionales.

Las imágenes dibujadas en la visualización son objetos abstractos que representan datos, esta representación usa los colores para mostrar información cuantitativa, tomando al color como un grado de libertad adicional; el modelado de colores esta basado en la combinación de diversos componentes básicos como en RGB, CMYK, HSV, HSL,YUV y YIC[Hodges, 2003].

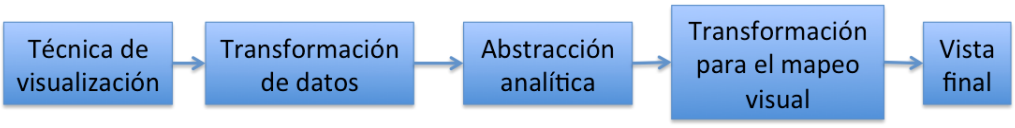


Figura 2.3: Modelo de referencia de estado de visualización de datos

Los datos representados son originados desde diversas fuentes, muchas de ellas proporcionan datos demasiado dispersos y requieren técnicas como la interpolación, con la finalidad de generar su representación gráfica. Operaciones estadísticas como media, varianza y desviación estándar son algunas de las transformaciones que sufren los datos antes de ser graficados. De modo que hay una relación entre la técnica de visualización, la transformación de datos, la abstracción analítica y la transformación para el mapeo visual que generan la vista final (ver figura 2.3).

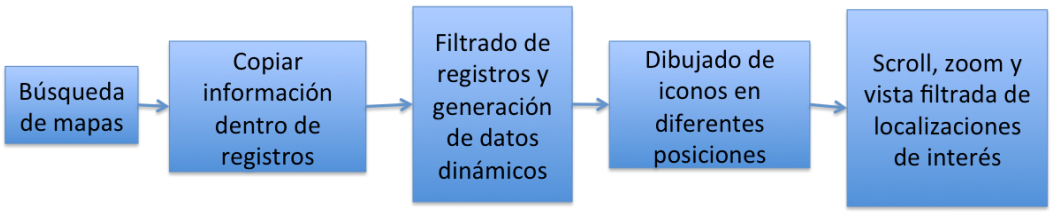


Figura 2.4: Modelo de estado de visualización de datos en una búsqueda de mapas

Por ejemplo en el despliegue de mapas, los datos iniciales son los mapas geográficos, las transformación de datos los convierte en registros, las abstracción analítica realiza los filtros dinámicos de los registros, el mapeo visual se encarga de dibujar los objetos en sus diferentes localizaciones y finalmente se obtiene la vista que permite realizar acciones como zoom, scroll y vista en puntos de interés[Chi, 2000]( ver figura 2.4).

Dentro del CinvesWall se desarrolló una aplicación de visualización científica, cuya finalidad es mostrar la correlación de Pearson de dos conjuntos de datos obtenidos de una base de datos relacional, la salida obtenida son modelos tridimensionales con diámetros definidos por la media, varianza y desviación estándar del conjunto de interés[Ramirez et al., 2012](

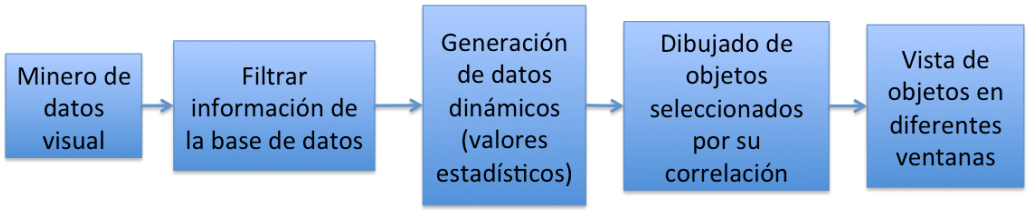


Figura 2.5: Modelo de estado de visualización de datos en el minero de datos visual

ver figura 2.5). Proyectos de minería de datos que trabajan sobre paredes de vídeo son Terascope [Zhang et al., 2003], Weka [Verta et al., 2005], iVici [Michnick and Tarassov, 2005], entre otros.

La graficación por computadora es la encargada de generar imágenes con apariencia realista, para eso se emplean diversas técnicas, incluyendo proyección en perspectiva, líneas ocultas y superficies removidas (perspective projection, hidden line, surface removal and stereographic images). La mayoría de estas técnicas trata de engañar a nuestro sistema visual, como la realidad virtual que emplea cascos para proporcionan control sobre el punto de vista. El sistema visual humano solo distingue 3 dimensiones por lo que es posible generar modelos semi realistas animados que lo pueden engañar[Wright, 2007].

### Render paralelo

Dentro de las tarjetas gráficas los encargados del procesamiento de datos son los *graphical processing unit* (GPUs). Estos procesadores cuentan con cierta cantidad de memoria para realizar el proceso de render de la imagen de forma paralela y una tarjeta gráfica actual tienen cientos de GPUs en su interior.

Cuando se realiza paralelismo, se particionan datos en porciones que pueden ser procesados independientemente por diferentes *processing elements* (PEs), actualmente estos PEs trabajan dentro de los GPUz[Foley et al., 1997].

El render paralelo permite escalar el desempeño de render por cada frame, de modo que hace posible visualizar grandes cantidades de datos. En este caso existen dos problemas principales (latencia y balanceo de cargas). Las primitivas que se pueden distribuir son los frames, los pixeles y los objetos.

- Frame distribution.- En este caso el procesamiento se realiza al frame completo de diferentes puntos de vista o momentos del tiempo. El render por cada frame puede mejorar la calidad de la imagen. Este enfoque permite escalar el desempeño pero no la distribución de los datos.

- Pixel distribution.- En este caso un conjunto de pixeles en el espacio de pantalla es distribuido en los GPUs como un proceso denominado *sort first rendering*. La distribución de lineas de pixeles da un buen balanceo de cargas pero no permite escalar

en la cantidad de datos. La distribución de mosaicos contiguos de pixeles permite escalar en la cantidad de datos pero tiene problemas con el cambio de perspectivas del objeto, esto se puede solucionar replicando y recargando datos dinámicamente con los cambios de vista. Dinámicamente el balanceo de cargas también necesita mantener la escala en el desempeño.

- Object distribution.- La distribución a nivel objeto se refiere a *sort-last rendering*, este proporciona buena escala de datos y puede proporcionar una buena escala en el desempeño, pero requiere imágenes intermedias de los nodos procesados para crear la imagen final.

### 2.2.2. Interacción Humano Computadora

En esta sección se abordan los temas relacionados a la interacción humano computadora (HCI), en la figura 2.6 se señala la parte principal de la arquitectura que requiere de estos temas.

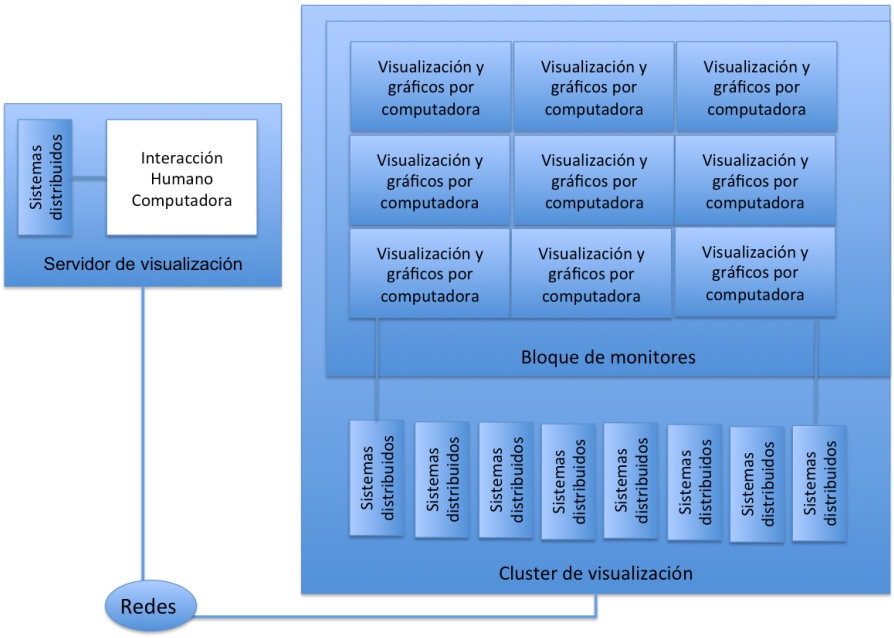


Figura 2.6: Conceptos relacionados con la interacción humano computadora

La interacción humano-computadora es el proceso que permite que un usuario pueda ver y manipular los datos del sistema, esta interacción se realiza mediante una interfaz. La interfaz es la representación gráfica de la aplicación, funciona como la encargada de convertir la entrada del usuario (eventos de usuario) en instrucciones que el sistema puede entender (ver figura 2.7).

Cuando un sistema tiene funcionalidad adecuada pero no es fácil de usar, los usuarios no obtienen la información necesaria para realizar sus tareas, eso se conoce como baja

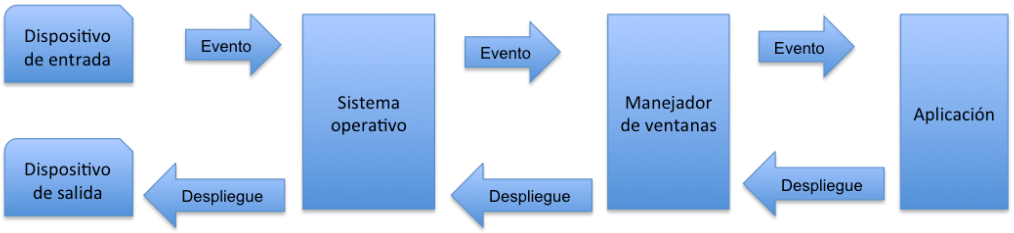


Figura 2.7: Interacción humano-computadora

usabilidad. Los factores que definen la usabilidad son los siguientes: desempeño, fácil de aprender, eficiencia de tareas, fácil de recordar, satisfacción subjetiva y fácil de entender [Lauesen, 2005][Myers, 1998]. El manejador de ventanas es un modelo que cumple con estos factores.

El manejador de ventanas es un modelo básico de programación que permite dibujar y actualizar la pantalla para aceptar entradas de usuario. Una de sus principales características es la habilidad de tener múltiples ventanas abiertas simultáneamente. Cada ventana puede desplegar diferentes aplicaciones o archivos que se han creado con una sola aplicación. El manejador de ventanas es parte del sistema operativo y se comunica directamente con él, ya que puede habilitar o desabilitar aplicaciones en ejecución, y manejar eventos.

El manejador de ventanas permite desarrollar interfaces de usuario mediante toolkits. [Myers et al., 2000], [Myers, 1995], [Shneiderman, 2000] han investigado y resumido algunos de los avances obtenidos en esta dirección. Por ejemplo *Interface builder* proporciona opciones para colocar componentes interactivos usando el mouse para crear ventanas [Robertson et al., 2000].

Una ventana es una porción rectangular de la pantalla que puede desplegar su contenido, por ejemplo un programa, iconos, texto, un archivo o una imagen, aparentando independencia del resto de la pantalla. Un icono es una pequeña imagen en una GUI que representa un programa, un archivo, un directorio o un dispositivo tal como un disco duro. Los iconos son usados en el escritorio y con los programas de las aplicaciones. Algunos ejemplos incluyen pequeños rectángulos que representan archivos, folders que representan directorios, un bote de basura representa un lugar donde se colocan archivos y directorios indeseables, etc.

Existen algunos principios para del desarrollo de GUIs[Norman, 1998]:

- El principio de mapeo.- Es un término técnico que significa la relación entre dos cosas, en este caso entre los movimientos realizados con los controles y el resultado esperado. El usuario debe identificar dos mapeos el efecto del control y la velocidad con la que trabaja el control. La ventaja del mapeo es que se aprende fácilmente y siempre se recuerda. Por ejemplo, la interacción mediante el mouse sobre algún icono o una ventana permite cambiar su posición dentro de la pantalla simulando el arrastre del objeto.

- El principio de retroalimentación.- La información de retroalimentación (al usuario acerca de las acciones que se han hecho) es un concepto conocido en la teoría de información y la ciencia del control. La GUI le proporciona al usuario retroalimentación visual inmediata sobre los efectos que tienen las acciones que realiza. Por ejemplo, cuando un usuario borra un icono que representa un archivo, el icono desaparece inmediatamente, confirmando que el archivo ha sido borrado o enviado a la basura.

Otro aspecto importante dentro del diseño del GUI es la ley de Fitt (describe el tiempo requerido para llegar a un objetivo con un movimiento), ya que es usada en la guía para el orden de los widgets, para reducir el tiempo de selección [MCGuffin, 2002], [Saund and Moran, 1995].

Las interfaces gráficas de usuario (GUI), manejan conceptos tales como widgets, toolkits y manejador de ventanas, dentro de un escritorio estándar. Los controles son conocidos como widgets, el widget incluye objetos de tipo menú, botones, barras desplazadoras y contenedores (canvas). Un objeto canvas es un widget que controla una subarea en el cliente donde es posible dibujar. En Mac OSX existe un objeto similar al objeto canvas pero que además proporciona características de manejo de eventos y jerarquización de objetos, este objeto se redefine de una clase abstracta llamada NSView[Inc., 2009], NSView es una clase que define el dibujo básico, manejo de eventos y arquitectura de despliegue en una aplicación. El NSView debe estar contenido en un objeto NSWindow, permitiendo el uso de una jerarquía de subviews. Un objeto view es el dueño de una superficie rectangular encontrada dentro de un superview, que es responsable de todos los eventos que ocurran dentro de él.

[Christian et al., 2009] [Arbab et al., 1992] [Coutinho et al., 2006] han creado esquemas diferentes para sus manejadores de ventanas. Dando como resultado un funcionamiento diferente como en el caso de Deskotheque, Manifold y Vitral. Tomando en cuenta las características más usuales de un manejador de ventanas podemos definir un esquema que muestre como funciona (ver figura 2.8). En este caso tenemos un contenedor que mantiene un arreglo de ventanas, cada ventana tiene información sobre su posición, su estado, su identificador y contenido. De esta manera se puede gestionar el comportamiento de cada ventana. Por ejemplo cuando ocurre un evento y se requiere saber el origen del mismo, es posible identificar la ventana por alguna de sus características como posición y estado.

### Manejo de eventos

Las interfaces gráficas son manejadas por los eventos de usuario, la mayoría de las aplicaciones gastan su tiempo esperando que un usuario diga que es lo siguiente que va a suceder. Además, una aplicación en ejecución puede recibir eventos que no se originen en la interfaz de usuario, tal como recepción de paquetes enviados por la red. Por ejemplo, en los manejadores de ventanas de cocoa[Nutting et al., 2009] y Qt [Blanchette and Summerfield, 2008], ambos tipos de eventos son convertidos en mensajes enviados a un objeto en la aplicación(NSEvent y QEvent).

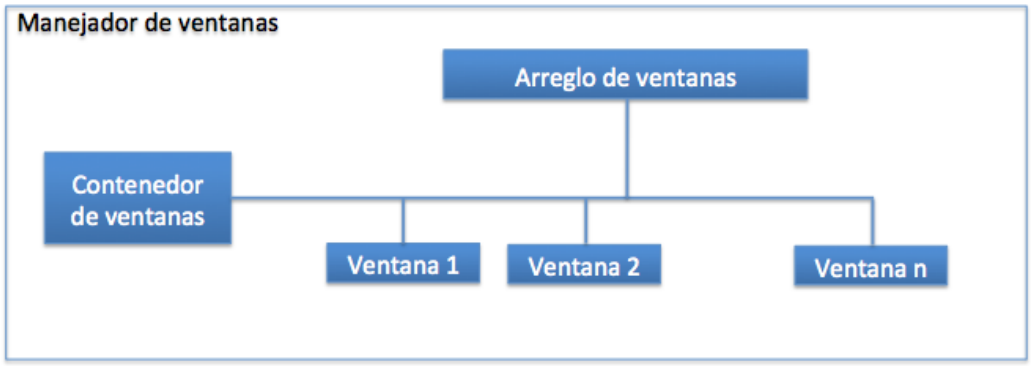


Figura 2.8: Manejo de ventanas

Cuando se utiliza el patrón de diseño basado en el modelo-vista-controlador, es la vista la encargada de recibir el evento como en el caso de Blender [Flavell, 2010], el evento generado debe ser reconocido mediante el manejador, y notificado para ejecutar la acción necesaria para el tipo de evento.

Los eventos se pueden representar por instancias de una clase. Un objeto de este tipo contiene información tal como la posición del evento en la ventana, y la hora en la que ocurrió, además aparece el tipo de dispositivo que lo inició. Existen muchos tipos de eventos pero los más comunes son los siguientes:

- Eventos de teclado.- Se genera cuando una tecla es presionada o liberada o cuando una tecla modificadora cambia. Se puede determinar el carácter o caracteres asociados al evento mediante el llamado al objeto que contiene el evento.

- Eventos de ratón.- Se genera por cambios en el estado del botón del ratón (pulsado o liberado) y durante el arrastre del ratón.

Un evento es una interrupción generada por algún dispositivo de entrada con la finalidad de realizar algún cambio sobre una aplicación, para que el evento sea recuperado de manera correcta se debe almacenar la información que lo distinga, como su posición, el dispositivo de origen, y el tiempo en el que ocurrió. Cuando el evento es reconocido por el sistema operativo generalmente lo almacena en una cola, como un objeto que contiene toda la información del evento. La cola con los eventos es revisada por las aplicaciones para obtener la información y transmitirla a su ventana.

La ventana se encarga de cambiar su visualización, dependiendo del tipo de evento ocurrido. Como el evento se inicia desde la visualización, se forma un ciclo (ver figura 2.9), este ciclo inicia y termina en la visualización final.

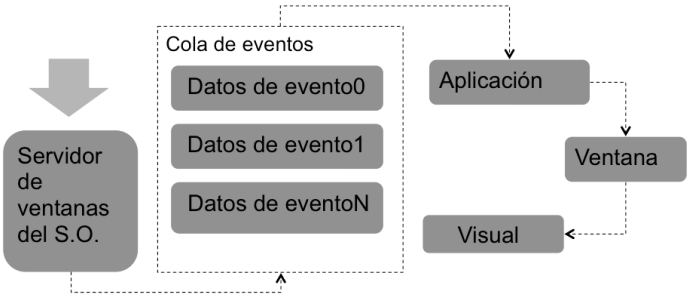


Figura 2.9: Ciclo del evento

### Patrones de diseño utilizados en la GUI

Un patrón de diseño es una solución flexible a problemas de diseño. En esta sección discutiremos patrones de diseño asociados a la GUI. Uno de los patrones de diseño más conocidos para el diseño de una GUI es el MVC. El patrón MVC se usa para construir interfaces de usuario, consiste de tres tipos de objetos. El modelo es el objeto aplicación, la vista es la presentación en pantalla, y el control define la reacción de la interfaz de usuario. MVC desacopla los objetos para incrementar flexibilidad y rehuso estableciendo un protocolo subscriptor/notificación entre ellos. Una vista debe asegurarse que la apariencia refleje el estado del modelo. Cuando los datos del modelo cambian, el modelo notifica a las vistas que dependen de él. En respuesta, cada vista obtiene una oportunidad para actualizarse. Esta aproximación permite adjuntar múltiples vistas a un modelo para proporcionar diferentes presentaciones. Se pueden crear nuevas vistas para un modelo sin re escribirlo[Krasner and Pope, 1988].

Otra característica del MVC es que la vista puede ser anidada. Por ejemplo, un control de botones puede implementarse como una vista compleja que contiene vistas de botones anidados. La interfaz de usuario para el objeto inspector puede consistir de vistas anidadas que pueden ser rehusadas. MVC permite el anidamiento de vistas mediante la clase CompositeView, una subclase de View. El objeto CompositeView actúa como el objeto View, puede ser usado como un View, pero también contiene y maneja vistas anidadas.

MVC permite cambiar el tipo de respuesta a la entrada del usuario sin cambiar su presentación visual. MVC encapsula el mecanismo de respuesta en el objeto control. Se usa una clase jerárquica de controladores, para facilitar la creación de nuevos controladores como una variación de los existentes [Freeman et al., 2004].

MVC usa otros patrones de diseño tal como Factory Method para especificar el controlador default para una vista y Decorator para agregar controles a una vista. Pero la relación principal del MVC esta dada por los patrones de diseño *Observer, Composite and Strategy*[Gamma et al., 1997].

Como ejemplo del patrón de diseño MVC (ver figura 2.10), se muestra un diagrama con clases del lado del cliente y el servidor, tenemos la clase de control (controller), las

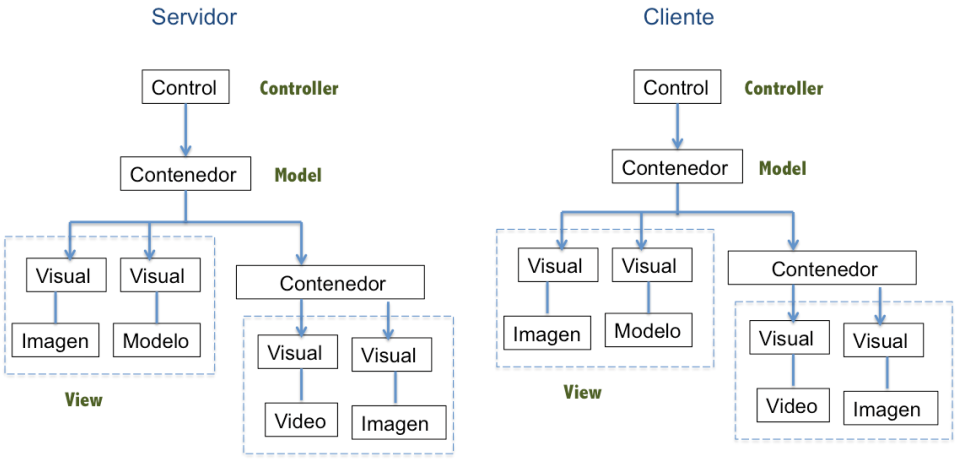


Figura 2.10: Patrón de diseño Model-View-Controller

clases encargadas del despliegue final (view) y el contenedor principal (model) que es quien almacena la estructura con la información que se despliega. En los siguientes patrones se mostrarán ejemplos similares sobre el mismo sistema.

### Patrón de diseño Composite

El patrón de diseño Composite emplea estructuras de árbol para representar partes de una jerarquía. Composite permite a los clientes tratar objetos individuales y composiciones del mismo modo. Esta definido para aplicaciones gráficas como editores y sistemas de captura esquemáticos, que le permiten a los usuarios construir complejos diagramas a partir de componentes simples. El usuario puede agrupar los componentes para formar grandes componentes, los cuales pueden ser agrupados de manera que formen un nuevo componente.

Este patrón se usa cuando se quiere representar partes de un todo con jerarquía de objetos como en el caso de las vistas del MVC. También se usa para permitir a los clientes habilitar o ignorar la diferencia entre composiciones de objetos y objetos individuales. Los clientes tratarán a todos los objetos en la composición como una estructura uniforme.

Los clientes usan la interfaz de la clase componente para interactuar con objetos en la estructura composite. Si el receptor es un componente interno, entonces la consulta es manejada directamente. Si el receptor es una composición, entonces usualmente se requieren operaciones adicionales para resolver la consulta[Hericko and Beloglavec, 2005].

Por ejemplo (ver figura 2.11), se muestra como podemos emplear el patrón de diseño Composite para tratar a cada objeto ya sea un objeto visual o un conjunto de objetos contenedores y objetos visuales como un solo objeto. De esta manera es posible manejar un elemento visual simple de la misma forma que a una ventana compuesta con diferentes objetos visuales. La definición de un objeto visual requiere un proceso que permita el

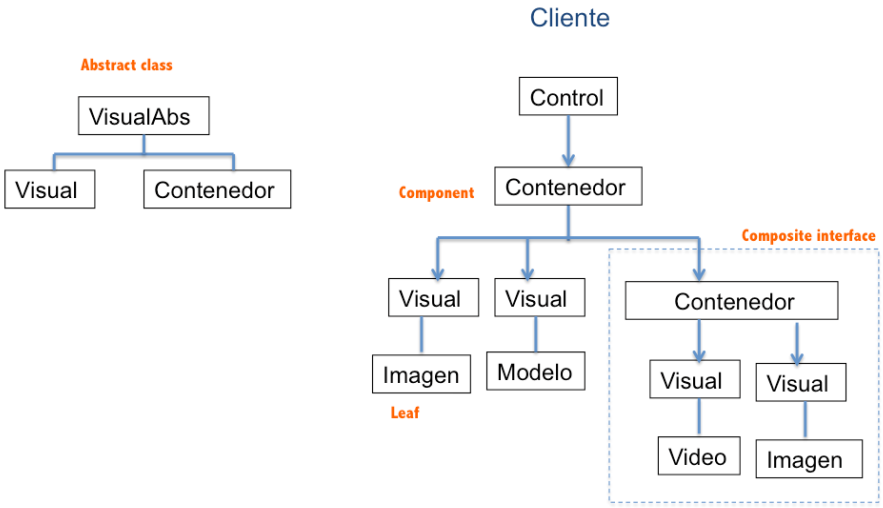


Figura 2.11: Patrón de diseño Composite

anidamiento del mismo, de manera que pueda tenerse un objeto más complejo, en este caso definimos 3 primitivas básicas, la imagen, el vídeo y un modelo 3D.

Posteriormente pueden ampliarse agregando Widgets, a fin de generar una ventana. El patrón de diseño Composite es usado en esta parte, ya que permite manejar a un conjunto de objetos compuestos como un objeto individual. Si tenemos un contenedor con dos visualizaciones y cambiamos su tamaño, este cambio se debe reflejar en todos los objetos contenidos. Cuando ocurra algún cambio en el cliente debe verse reflejado en el servidor, y al mantener el mismo patrón del lado de los servidores se facilita la manipulación del conjunto de objetos al enviar la referencia de cambio a un solo objeto compuesto. En el capítulo de Framework se mencionará la estrategia adoptada para su implementación (uso de la clase abstracta VisualAbs).

### Patrón de diseño Observer (Publish-Subscribe)

El patrón de diseño Observer define una dependencia uno a muchos entre objetos de modo que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente.

El patrón de diseño Observer, se emplea en MVC para que las vistas se enteren cuando ocurra un cambio en el modelo. De modo que las vistas se puedan actualizar con los datos del modelo actual. Este diseño puede ser aplicado de modo general , desacoplando objetos donde el cambio de un objeto afecte a un conjunto de objetos.

Cuando particionamos un sistema en una colección de clases, requerimos mantener consistencia entre los objetos relacionados. Para lograr que esta consistencia no obligue a

tener clases estrechamente acopladas, se tienen clases definidas con datos de la aplicación y clases con las interfaces de la aplicación, de esa manera las clases pueden ser rehusadas independientemente.

El patrón Observer describe como establecer las relaciones entre los observadores y los objetivos. Un objetivo puede tener un número de observadores dependientes. Todos los observadores son notificados cuando el objetivo cambia su estado, en respuesta cada observador consultará al objetivo para sincronizar su estado con el estado del objetivo.

Todos los objetivos saben si tienen una lista de observadores, cada uno conforma una simple interfaz de la clase abstracta de los observadores. Los objetivos no conocen las clases concretas de los observadores. De modo que se tiene una desacoplación entre objetivos y observadores de forma abstracta y mínima.

Debido a que los objetivos y los observadores no están acoplados, pueden permanecer en diferentes capas de abstracción del sistema. El tipo de comunicación empleado por el patrón Observer permite la notificación automática usando Broadcast.

Cuando ocurre una notificación los observadores se ven forzados a trabajar duro para deducir los cambios ocurridos en el objetivo, ya que los protocolos no proporcionan detalles de los cambios ocurridos en el objetivo[Austrem, 2008].

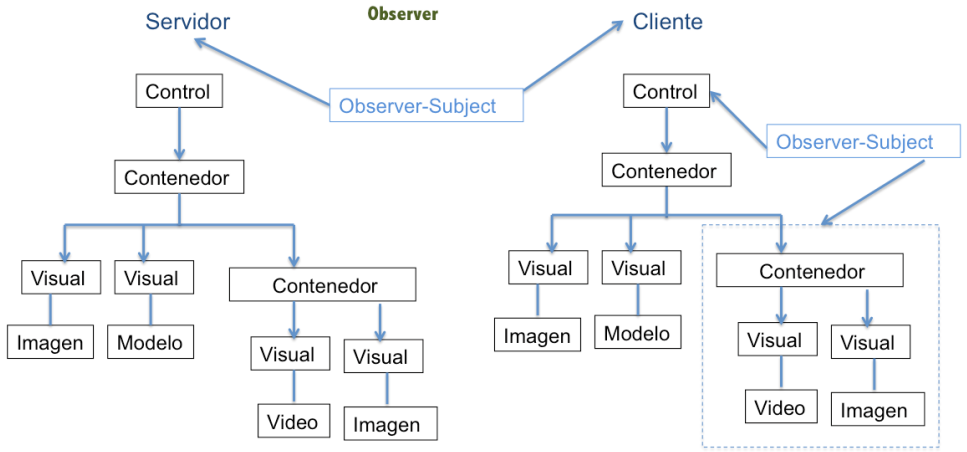


Figura 2.12: Patrón de diseño Observer

Por ejemplo en las clases antes mencionadas (ver figura 2.12), se requiere obtener información sobre lo que sucede directamente en los objetos de despliegue final (objeto visual). Cuando ocurre algún cambio dentro de los objetos visuales debe ser informado a los contenedores, esto se puede hacer mediante el patrón de diseño Observer, además podemos convertir a los Servidores en observadores para ser informados por los cambios en el cliente. De manera que este patrón es empleado de manera local y de manera remota para informar los cambios generados principalmente por los eventos de usuario

### Patrón de diseño State

El patrón de diseño State permite a un objeto alterar su comportamiento cuando ocurran cambios internos de estado. Por ejemplo una clase que represente una conexión de red. Un objeto puede estar en uno de muchos estados diferentes: establecido, escuchando, cerrado. Cuando el objeto recibe peticiones de otros objetos, responde de acuerdo a su estado actual. La idea principal de este patrón es introducir una clase llamada State que representa los estados de la conexión de la red. La clase State implementa un comportamiento específico por estado.

Este patrón es empleado cuando el comportamiento del objeto depende de su estado, y estos cambios ocurren en tiempo de ejecución. En algunos casos operaciones muy grandes tienen algunas partes condicionales que dependen del estado del objeto, este estado se representa por una o más constantes. Cuando estas operaciones contienen la misma estructura condicional, se puede usar el patrón State para mantener a cada condición en clases separadas, de manera que el estado del objeto se pueda tratar como un objeto propio.

El patrón de estados coloca todos los comportamientos asociados con un estado particular en un objeto. Debido a que todo el código de estados específicos están en una subclase State, nuevos estados y transiciones pueden ser agregados fácilmente para definir nuevas subclases.

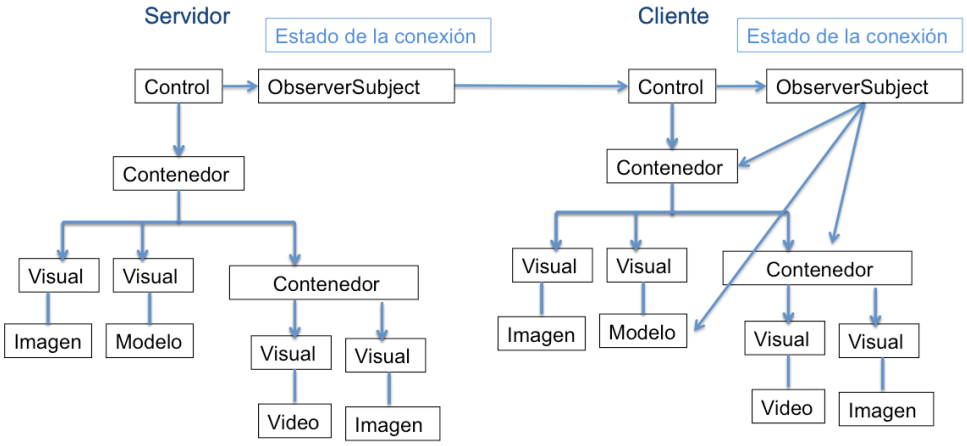


Figura 2.13: Patrón de diseño State

El ejemplo común de este patrón de diseño es el estado de la conexión de red (ver figura 2.13), en nuestro diagrama de clases mostramos al control como encargado de mantener el conocimiento del estado de la red, empleando este objeto es posible que el control delegue esta responsabilidad a un objeto comunicación, que pueda funcionar en los diferentes estados de la conexión.

### Patrón de diseño Strategy (Policy)

El patrón de diseño Strategy define una familia de algoritmos, los encapsula y los hace intercambiables. Strategy permite que los algoritmos varíen independientemente de los clientes que los usen. Este patrón permite emplear muchos algoritmos, sin necesidad de incluir el código de cada uno. De esta manera no se hace tan complejo su uso y su mantenimiento, esto es posible ya que dichos algoritmos son apropiados en diferentes tiempos.

Además se facilita la adición de nuevos algoritmos, esto se hace mediante la encapsulación de algoritmos dentro de clases, cada algoritmos encapsulado se conoce como Strategy.

Este patrón de diseño es usado cuando muchas clases relacionadas difieren sólo por su comportamiento, Strategy proporciona una manera para configurar una clase con uno de muchos comportamientos. De esta manera permite variar los algoritmos. También es posible usar Strategy para evitar la exposición de datos usados en un algoritmo.

Encapsular el algoritmo en clases Strategy permite variar el algoritmo independientemente de su contexto, facilitando su manutención, extensión y comprensión.

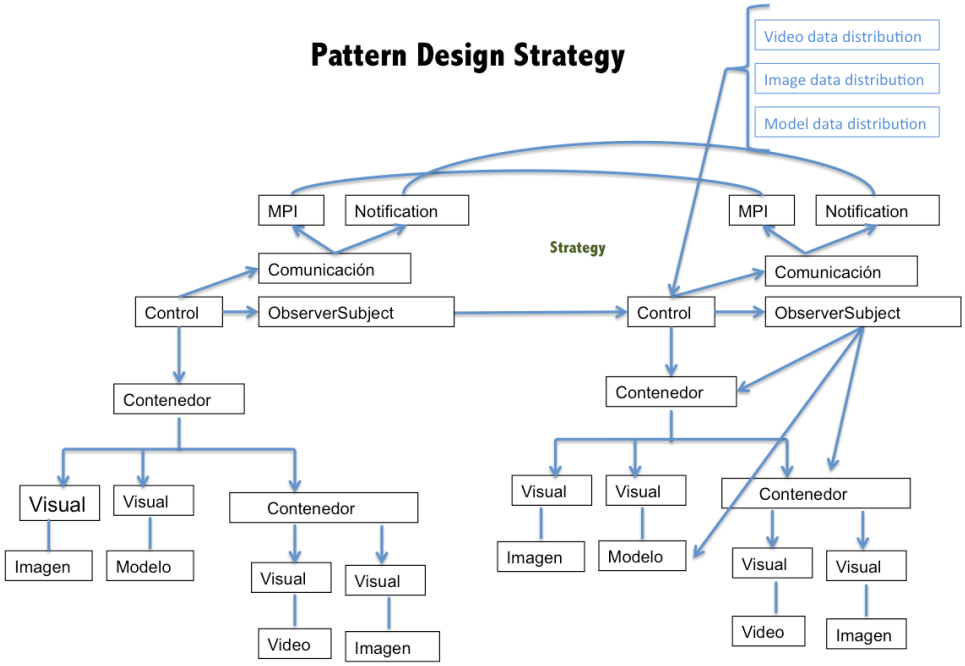


Figura 2.14: Patrón de diseño Strategy

Algunos algoritmos deben variar con respecto al tipo de información que se este tratando, en este caso definimos tres básicas, y para cada una existe un algoritmo de distribución diferente (ver figura 2.14). Se puede emplear el patrón de diseño Strategy para mantener los algoritmos dentro de objetos empleados cuando se requieran. En este caso las decisiones se toman de manera dinámica ya que la decisión del tipo de objeto depende de la

entrada del usuario. En el caso del envío de información entre el cliente y servidor también tenemos al menos dos variantes, si es control o si es un dato inicial.

En conclusión podemos utilizar estos patrones de diseño para obtener el modelo que nos permita manejar la pared de vídeo, utilizando la estructura del modelo Model-View-Control. La cual utiliza diversos patrones de diseño como Strategy, Observer y State.

### 2.2.3. Programación orientada a objetos (sistemas distribuidos)

Los objetos distribuidos son los componentes que permiten la comunicación remota con los nodos del cluster de visualización. Por lo que lo primero que debemos identificar es el funcionamiento del paradigma orientado a objetos, para posteriormente mostrar el funcionamiento de los objetos distribuidos 2.15. Cabe resaltar que los objetos distribuidos son parte esencial del modelo DVO.

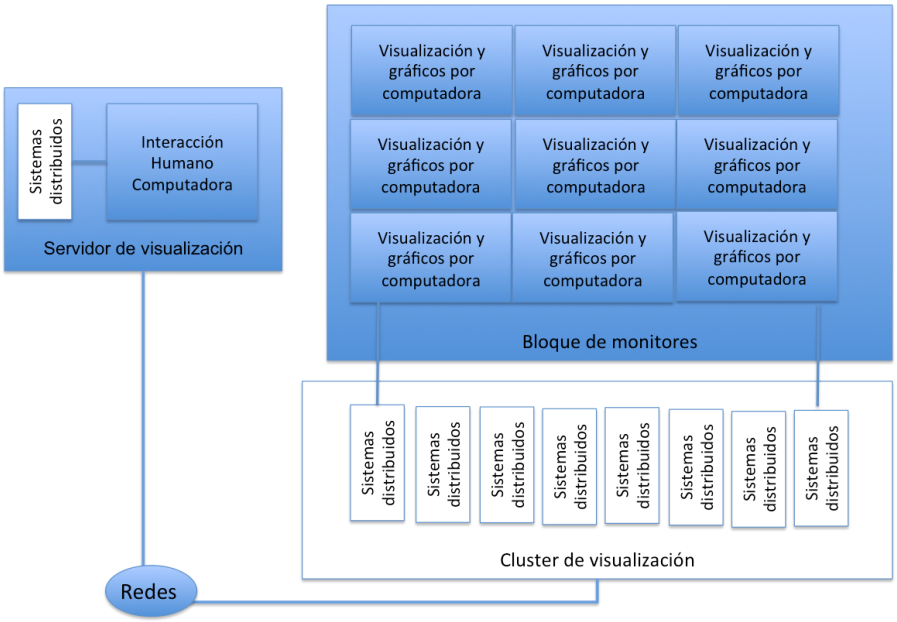


Figura 2.15: Conceptos relacionados a los sistemas distribuidos

Un *modelo* es una representación sistemática del dominio de un problema que permita o facilite su estudio. El *modelo* describe cuales son los principios que se usan para producir esta representación. En el *modelo* se incluyen los conceptos que reflejan la filosofía para la aproximación al área del problema. Cada *modelo* define conceptos que se enfocan en aspectos específicos del área del problema que se modelará. El concepto principal del *modelo de objeto* es el *objeto*. Un *objeto* se obtiene de la descomposición de un dominio de un problema, por lo que se convierte en un contenedor de una parte de un problema. La parte del problema elegida es relativa a un observador y su aproximación al área del problema determina su nivel de *abstracción*. Todos los *objetos* tienen definida una

*interfaz*, la *interfaz* distingue a un *objeto* de los demás. Los *objetos* tienen *estados* que los caracterizan, estos *estados* usualmente son una influencia en su comportamiento. De esta manera el estado y el comportamiento del objeto definen la funcionalidad que tiene en su entorno [Puder et al., 2006].

El principio de un modelo de objeto permite una representación sistemática del área del problema en base a los conceptos de abstracción, modularidad, encapsulación y herencia.

### Objeto distribuido

Un sistema distribuido tiene una estructura compleja, que requiere una organización precisa en sus componentes de software para su correcto funcionamiento. Es posible hacer una separación entre la organización lógica y física de los componentes del sistema. Una arquitectura de software que permite una organización lógica con bastante libertad es la arquitectura basada en objetos. Dentro de esta arquitectura cada objeto es un componente, estos componentes se comunican entre ellos mediante ciertos mecanismos [Tanenbaum and Steen, 2008], [Coulouris et al., 2007].

La característica fundamental de un objeto es la clara separación que hay entre interfaz e implementación. Esta separación es crucial dentro de un sistema distribuido. Gracias a esta separación es posible colocar las interfaces en una máquina y las implementaciones de los objetos en otras. Esta organización comúnmente se conoce como *objeto distribuido*.

Existe una capa en donde estos objetos son implementados de modo que el acceso ocurra a través de cualquier plataforma, proporcionando mayor eficiencia y facilidad en su manejo. Esta capa se encuentra entre las aplicaciones y el sistema operativo y comúnmente es conocida como *middleware* (ver figura 2.16).

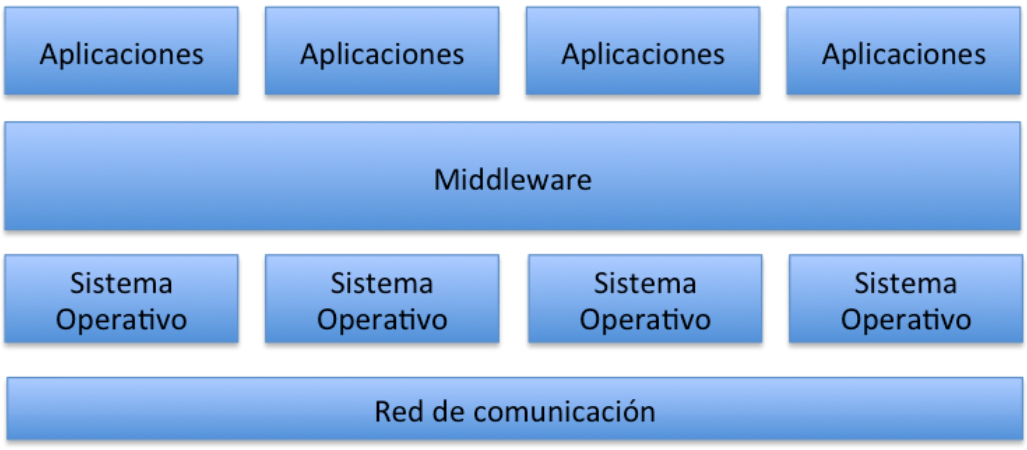


Figura 2.16: Arquitectura middleware

Algunos de los modelos propuestos para la implementación de objetos distribuidos son los siguientes:

- El modelo de objetos fragmentados, en el cual un objeto puede ser separado en muchas partes, localizadas en diferentes nodos que cooperan entre sí para permitir la funcionalidad del objeto. Un ejemplo de un sistema que usa este modelo es Globe [van Steen M. et al., 2002].

- El modelo de replicación de objetos, en el cual muchas copias o replicas de un objeto dado pueden coexistir. La motivación para replicar objetos es incrementar la disponibilidad y el desempeño. Las replicas de un objeto deben ser consistentes.

- El modelo de migración de objetos, en el cual un objeto puede moverse de un nodo a otro. La movilidad de los objetos es usada para mejorar el desempeño a través del balanceo de cargas, y la adaptación dinámica de aplicaciones con ambientes variables.

Estos modelos pueden ser combinados, los objetos fragmentados pueden ser replicados, etc. Una aplicación distribuida usando objetos remotos es ejecutada como un conjunto de procesos localizados en los nodos de una red.

Las ventajas del uso de objetos distribuidos son las siguientes:

- El diseño de aplicaciones puede tomar ventajas de las expresiones, abstracciones y flexibilidad de un modelo de objeto.

- El encapsulado de la implementación de un objeto debe ser localizada en algún sitio, el cuál puede ser especificado de acuerdo a un criterio, por ejemplo la restricción de la administración o la seguridad del sistema.

- La escalabilidad se emplea para potenciar el proceso de distribución sobre redes de servidores, el cual puede ser extendido para incrementar la disponibilidad del sistema.

### Implementaciones de Objetos Distribuidos

CORBA es un middleware basado en un modelo de objeto descrito en la arquitectura de manejo de objeto (OMA). El OMA (Object Management Architecture) describe una plataforma general para el desarrollo distribuido de aplicaciones orientadas a objeto. El Common Object Request Broker Architecture (CORBA)(ver figura 2.17), es una especialización de OMA. Las principales características de OMA son una abstracción del modelo de objeto y una arquitectura de referencia. El modelo de objeto de OMA hace una diferencia entre la semántica del objeto y la implementación del objeto. La semántica del objeto describe las características que son visibles a los clientes y la implementación detalla los conceptos necesarios para la ejecución del objeto[Object Management Group, 2011].

RMI significa (Remote Method Invocation) (ver figura 2.18), es una llamada a un método en un objeto remoto, la idea de RMI es lograr la invocación de un método de manera independiente a la JVM (Java Virtual Machine). Las capas de red dentro de RMI funcionan de la siguiente manera. La capa de transporte está basada en TCP/IP. La capa

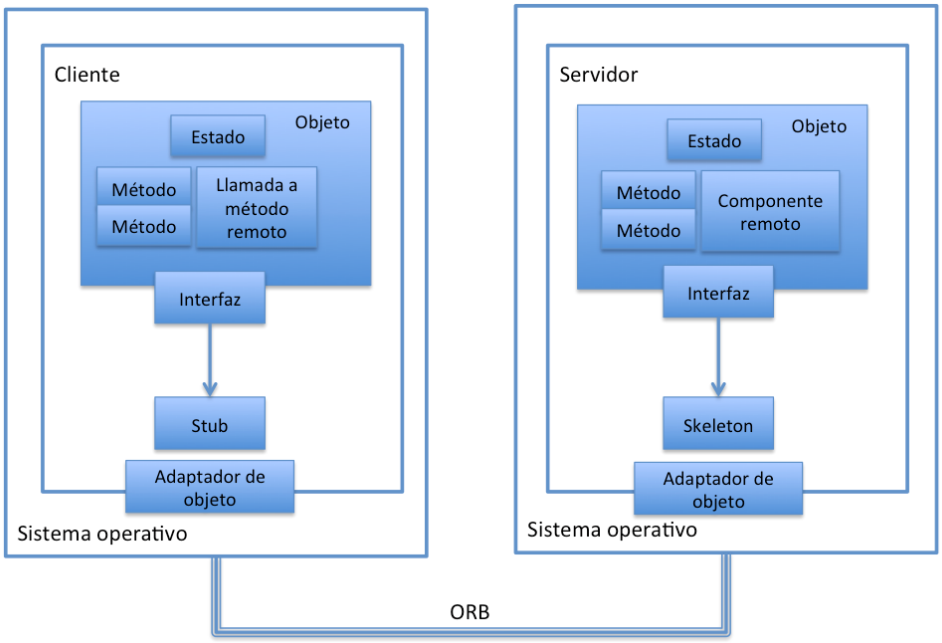


Figura 2.17: Modelo de objeto distribuido de CORBA.

de sesión se conecta con los clientes de los objetos remotos mediante una referencia uno a uno. La capa de presentación, es la capa que traduce los datos de sesión de la capa de aplicación y viceversa. La capa de aplicación del cliente traduce su invocación a una llamada al objeto Stub, que se traduce a JRMP, después a TCP, hasta llegar al nivel de hardware del cliente, que la envía al hardware donde está el objeto remoto. Aquí se recorre el orden inverso, desde el hardware hasta la capa de aplicación. Esta arquitectura mantiene la flexibilidad. Podemos reemplazar una sin afectar la eficacia del conjunto. Es decir, se puede usar UDP( User Datagram Protocol) en lugar de TCP[Grosso, 2002].

En la implementación de Objetive-C los objetos distribuidos tienen un proceso servidor, que se hace público, y un objeto cliente el cuál se puede conectar al proceso servidor. Una vez que la conexión se hizo, el proceso cliente invoca uno de los métodos del objeto servidor como si este método existiera en el proceso cliente, la sintaxis no cambia. El sistema en tiempo de ejecución realiza los procesos necesarios para la transmisión de los datos entre los procesos (ver figura 2.19 ).

### 2.2.4. Redes

En esta sección se mencionan diversos protocolos de redes para envío de mensajes, en la figura 2.15 se señala la parte principal de la arquitectura que requiere de estos protocolos.

El modelo de interconexión de sistemas abiertos (ISO/IEC 7498-1), también llamado OSI (en inglés open system interconnection) es el modelo de red descriptivo creado por

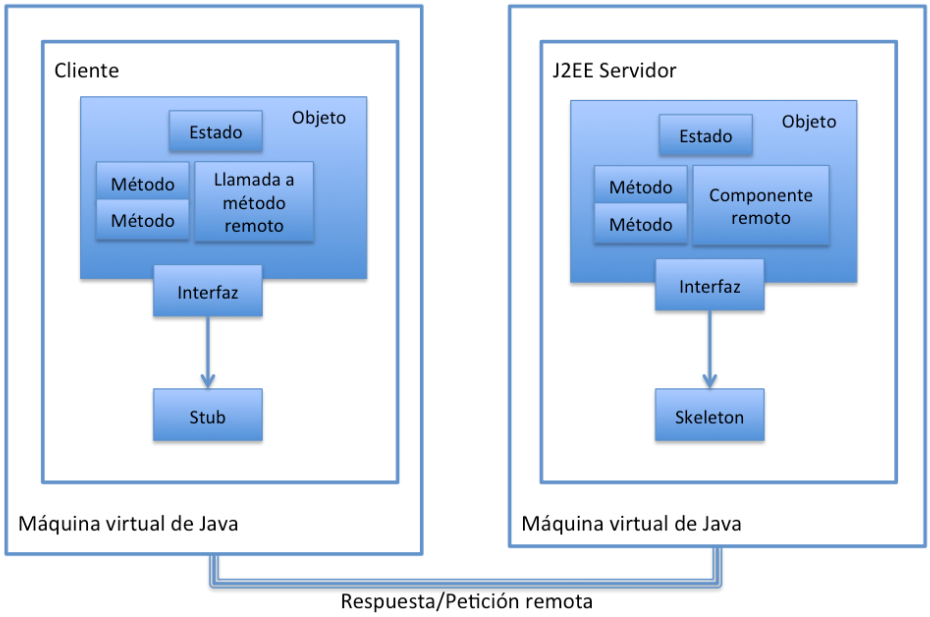


Figura 2.18: Modelo de objeto distribuido de JAVA

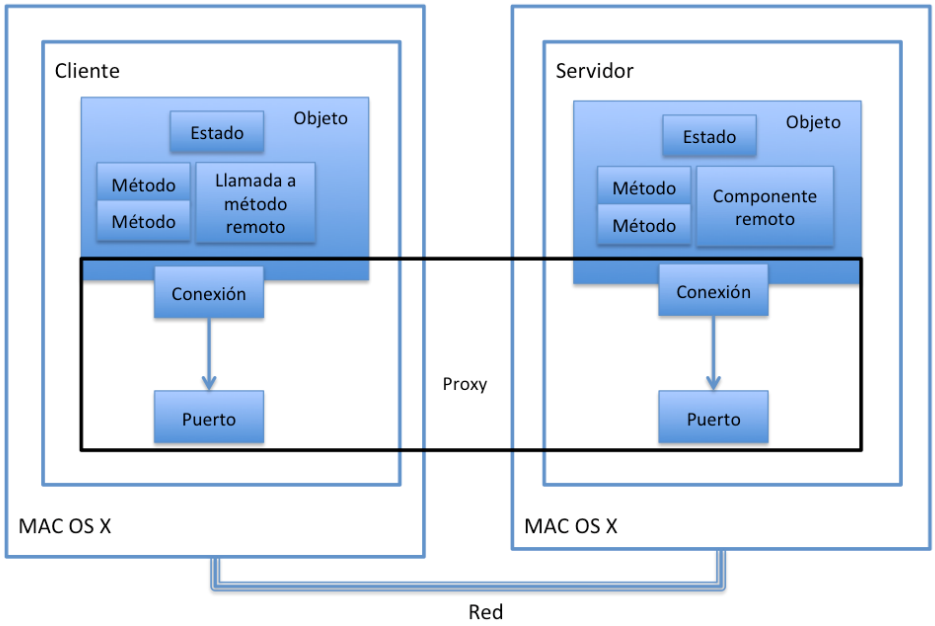


Figura 2.19: Modelo de objeto distribuido de Objetive-C

la Organización Internacional para la Estandarización (ISO)[Zimmermann, 1980]. La ISO tiene un modelo de 7 capas que permiten la transmisión de mensajes en red, dentro de

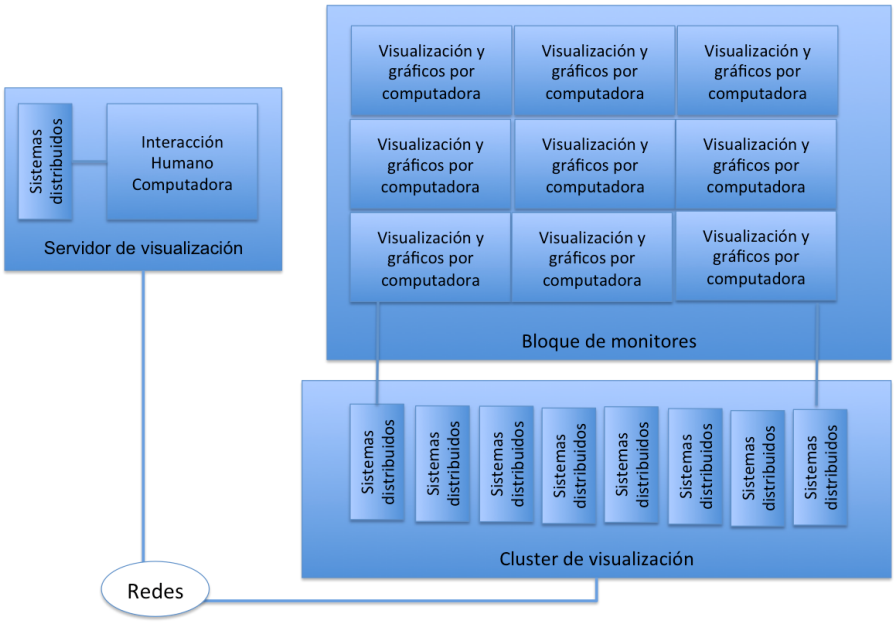


Figura 2.20: Conceptos relacionados a las redes

cada capa se puede utilizar alguno de los diferentes protocolos existentes, a continuación se muestran los protocolos encontrados para cada capa, con la finalidad de proporcionar opciones para la implementación de la capa de comunicación del modelo DVO.

- La capa de aplicación.- Esta capa define el intercambio de mensajes entre las aplicaciones, algunos protocolos de esta capa son FTP,DNS, DHCP, HTTP, POP, SMTP, SSH, TELNET, TFTP, etc.

- La capa de presentación.- La capa de presentación puede considerarse el traductor del modelo OSI. Esta capa toma los paquetes de la capa de aplicación y los convierte a un formato genérico que pueden leer todas las computadoras. Por ejemplo, los datos escritos en caracteres ASCII se traducirán a un formato más básico y genérico. También se encarga de cifrar los datos así como de comprimirlos para reducir su tamaño. El paquete que crea la capa de presentación contiene los datos prácticamente con el formato con el que viajarán por las restantes capas de la pila OSI (aunque las capas siguientes Irán añadiendo elementos al paquete. Esta capa tiene la misión de coger los datos que han sido entregados por la capa de aplicación, y convertirlos en un formato estándar que otras capas puedan entender. En esta capa tenemos como ejemplo los formatos MP3, MPG, GIF, etc.

- La capa de sesión.- La capa de sesión establece, administra y termina las sesiones entre las aplicaciones. Esto incluye el inicio, terminación y la re sincronización de dos computadoras que están manteniendo una sesión, algunos son NFS, SQL, RPC, DNA SCP y NetBIOS.

- La capa de transporte.- Esta capa se encarga de la transferencia de datos, el control de flujo y de congestión, los protocolos usados comúnmente en esta capa son TCP y UDP.

  - UDP (protocolo de datagramas de usuario).- UDP es un protocolo de transporte no orientado a la conexión. Este protocolo proporciona una forma para que las aplicaciones envíen datagramas IP encapsulados sin tener una conexión.
  - TCP (protocolo de control de transmisión).- TCP se diseñó específicamente para proporcionar un flujo de bytes confiable de extremo a extremo a través de una interred no confiable. Una interred difiere de una sola red debido a que diversas partes podrían tener diferentes topologías, anchos de banda, retardos, tamaños de paquete. TCP tiene un diseño que se adapta de manera dinámica a las propiedades de la interred y que se sobrepone a muchos tipos de situaciones.
  - SCTP.- Stream Control Transmission Protocol (SCTP) es un protocolo de comunicación de capa de transporte que fue definido por el grupo SIGTRAN de IETF en el año 2000. El protocolo está especificado en la RFC 2960, y la RFC 3286 brinda una introducción al mismo. SCTP es una alternativa a los protocolos de transporte TCP y UDP ya que permite tener confiabilidad, control de flujo y secuenciación como TCP. Sin embargo, SCTP opcionalmente permite el envío de mensajes fuera de orden y a diferencia de TCP, SCTP es un protocolo orientado al mensaje (similar al envío de datagramas UDP).
  - RTP.- RTP es un protocolo de transporte de tiempo real, este protocolo esta diseñado para la transmisión de datos de tipo multimedia, tal como audio y vídeo. Este protocolo proporciona alta velocidad en conectividad.

- La capa de red.- En esta etapa se genera el paquete y se determina su ruta desde la fuente hasta el destino, algunos protocolos de esta capa son los siguientes:

  - IP (IPv4, IPv6, IPsec) Internet Protocol es un protocolo de comunicación de datos digitales. Su función principal es el uso bidireccional en origen o destino de comunicación para transmitir datos mediante un protocolo no orientado a conexión que transfiere paquetes conmutados a través de distintas redes físicas previamente enlazadas según la norma OSI de enlace de datos.
  - OSPF Open Shortest Path First (frecuentemente abreviado OSPF) es un protocolo de enrutamiento jerárquico de pasarela interior o IGP (Interior Gateway Protocol), que usa el algoritmo Dijkstra enlace-estado (LSA - Link State Algorithm) para calcular la ruta más corta posible. Construye una base de datos enlace-estado (link-state database, LSDB) idéntica en todos los enrutadores de la zona. OSPF no usa ni TCP ni UDP, sino que usa IP directamente, mediante el protocolo IP 89.
  - IS-IS El protocolo IS-IS es un protocolo de estado de enlace, o SPF (shortest path first), por lo cual, básicamente maneja una especie de mapa con el que se

fabrica a medida que converge la red. Fue creado con el fin de crear un acompañamiento a CNS (Protocol for providing the Connectionless-mode Network Service). Para poder soportar dominios grandes, la previsión está hecha para que el ruteo intradominio sea organizado jerárquicamente. Un dominio grande puede ser dividido administrativamente en áreas. Soportan máscaras de subred de diferente longitud, puede usar multicast para encontrar routers vecinos mediante paquetes hello y pueden soportar autentificación de actualizaciones de encaminamiento.

- RIP RIP son las siglas de Routing Information Protocol (Protocolo de Información de Enrutamiento). Es un protocolo de puerta de enlace interna o IGP (Internal Gateway Protocol) utilizado por los routers (encaminadores), aunque también pueden actuar en equipos, para intercambiar información acerca de redes IP. Es un protocolo de Vector de distancias ya que mide el número de *saltos* como métrica hasta alcanzar la red de destino. El límite máximo de saltos en RIP es de 15, 16 se considera una ruta inalcanzable o no deseable.

- ICMP, ICMPv6 El Protocolo de Mensajes de Control de Internet o ICMP (por sus siglas en inglés de Internet Control Message Protocol) es el sub protocolo de control y notificación de errores del Protocolo de Internet (IP). Como tal, se usa para enviar mensajes de error, indicando por ejemplo que un servicio determinado no está disponible o que un router o host no puede ser localizado. ICMP no se utiliza directamente por las aplicaciones de usuario en la red. Los mensajes ICMP son comúnmente generados en respuesta a errores en los datagramas de IP o para diagnóstico y ruteo.

- IGMP El protocolo de red IGMP se utiliza para intercambiar información acerca del estado de pertenencia entre enrutadores IP que admiten la multidifusión y miembros de grupos de multidifusión. Los hosts miembros individuales informan acerca de la pertenencia de hosts al grupo de multidifusión y los enrutadores de multidifusión sondean periódicamente el estado de la pertenencia. Todos los mensajes IGMP se transmiten en datagramas IP.

- DHCP DHCP (sigla en inglés de Dynamic Host Configuration Protocol) es un protocolo de red que permite a los clientes de una red IP obtener sus parámetros de configuración automáticamente. Se trata de un protocolo de tipo cliente/servidor en el que generalmente un servidor tiene una lista de direcciones IP dinámicas y las va asignando a los clientes conforme éstas van estando libres, sabiendo en todo momento quién ha estado en posesión de esa IP, cuánto tiempo la ha tenido y a quién se la ha asignado después.

- BOOTP son las siglas de Bootstrap Protocol. Es un protocolo de red UDP utilizado por los clientes de red para obtener su dirección IP automáticamente. Normalmente se realiza en el proceso de arranque de los ordenadores o del sistema operativo. Este protocolo permite a los ordenadores sin disco obtener una dirección IP antes de cargar un sistema operativo avanzado. Históricamente ha

sido utilizado por las estaciones de trabajo sin disco basadas en UNIX (las cuales también obtenían la localización de su imagen de arranque mediante este protocolo) y también por empresas para introducir una instalación preconfigurada de Windows en PC recién comprados (típicamente en un entorno de red Windows NT).

- La capa de enlace de datos

  - Ethernet o IEEE 802.3 Ethernet es un estándar de redes de área local para computadoras con acceso al medio por contienda CSMA/CD. CSMA/CD (Acceso Múltiple por Detección de Portadora con Detección de Colisiones), es una técnica usada en redes Ethernet para mejorar sus prestaciones. Ethernet define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos del modelo OSI.
  - IEEE 802.11 o Wi-Fi es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Los dispositivos habilitados con Wi-Fi tal como un ordenador personal, una consola de vídeojuegos, un smartphone o un reproductor de audio digital pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Dicho punto de acceso (o hotspot) tiene un alcance de unos 20 metros (65 pies) en interiores y al aire libre una distancia mayor.
  - PPP (Point to point protocol o protocolo punto a punto) Point-to-point Protocol (en español Protocolo punto a punto), también conocido por su acrónimo PPP, es un protocolo de nivel de enlace estandarizado en el documento RFC 1661. Por tanto, se trata de un protocolo asociado a la pila TCP/IP de uso en Internet.
  - HDLC (High level data link control o protocolo de enlace de alto nivel) es un protocolo de comunicaciones de propósito general punto a punto y multipunto que opera a nivel de enlace de datos. Se basa en ISO 3309 e ISO 4335. Surge como una evolución del anterior SDLC. Proporciona recuperación de errores en caso de pérdida de paquetes de datos, fallos de secuencia y otros, por lo que ofrece una comunicación confiable entre el transmisor y el receptor.

- La capa física.- Los protocolos de nivel físico son: V.92, xDSL, IrDA, 10BASE-T, 10BASE2, 10BASE5, 100BASE-TX, 100BASE-FX, 100BASE-T, 1000BASE-T, 1000BASE-SX y otras. 1000Base-T es un estándar para redes de área local del tipo Gigabit Ethernet sobre cable de cobre trenzado, emplea todos los cuatro pares de hilos del cable, transmitiendo simultáneamente en ambos sentidos y por cada uno de ellos. Se multiplica así por ocho la velocidad de modulación, a costa de aplicar un sistema electrónico de cancelación de eco. Puede funcionar sobre cable de categoría 5 mejorado (UTP 5e) o superior. Emplea una modulación por amplitud de pulsos con señales de 5 niveles denominada PAM-5 para alcanzar la velocidad de 1 Gb/s en modo full duplex.

**Comparaciones entre protocolos de la misma capa**

Existen diferencias importantes en el modo de operar de IS-IS y OSPF, por ejemplo, en el modo en que la dirección de área es asignada. En IS-IS, la dirección de área y de host son asignados al router entero, mientras que en OSPF (Open Shortest Path First) el direccionamiento es asignado al nivel de interfaz.

Es importante también la diferencia entre estos protocolos al manejar los paquetes hello. Este es el único método por el cual los routers pueden saber si un router vecino sigue estando disponible en la red. A diferencia de OSPF, los routers IS-IS son capaces de enviar dos tipos diferentes de saludos (paquetes hello). Los routers IS-IS pueden ser de Nivel 1, Nivel2 o Nivel 1-2, los routers CISCO son routers L1-L2, por lo que cada interfaz IS-IS estará habilitada para enviar tanto mensajes hello L1 como L2.

El protocolo SCTP tiene capacidad de Multihoming, en la cual uno (o dos) de los extremos de una asociación (conexión) pueden tener más de una dirección IP. Esto permite reaccionar en forma transparente ante fallos en la red. Entrega de los datos en trozos que forman parte de flujos independientes y paralelos —eliminando así el problema de head of the line blocking que sufre TCP. Es capaz de seleccionar y monitorizar caminos, seleccionando un camino primario y verificando constantemente la conectividad de cada uno de los caminos alternativos. Mecanismos de validación y asentimiento como protección ante ataques por inundación, permitiendo notificación de trozos de datos duplicados o perdidos. SCTP fue diseñado inicialmente por el grupo Sigtran para transportar señalización telefónica SS7 sobre IP. La intención fue la de proporcionar en IP algunas de las características de confiabilidad de SS7. Por su versatilidad luego se ha propuesto utilizarlo en otras áreas, como por ejemplo para transportar mensajes de los protocolos DIAMETER o SIP.

UDP proporciona un nivel de transporte no fiable de datagramas, ya que apenas añade la información necesaria para la comunicación extremo a extremo al paquete que envía al nivel inferior. Lo utilizan aplicaciones como NFS (Network File System) y RCP (comando para copiar ficheros entre ordenadores remotos), pero sobre todo se emplea en tareas de control y en la transmisión de audio y vídeo a través de una red. Esto es porque no hay tiempo para enviar de nuevo paquetes perdidos cuando se está escuchando a alguien o viendo un vídeo en tiempo real.

No introduce retardos para establecer una conexión, no mantiene estado de conexión alguno y no realiza seguimiento de estos parámetros. Así, un servidor dedicado a una aplicación particular puede soportar más clientes activos cuando la aplicación corre sobre UDP en lugar de TCP.

TCP es el protocolo que proporciona un transporte fiable de flujo de bits entre aplicaciones. Está pensado para poder enviar grandes cantidades de información de forma fiable, liberando al programador de la dificultad de gestionar la fiabilidad de la conexión (retransmisiones, pérdida de paquetes, orden en el que llegan los paquetes, paquetes duplicados) que gestiona el propio protocolo. Pero la complejidad de la gestión de la fiabilidad tiene un costo en eficiencia, ya que para llevar a cabo las gestiones anteriores se tiene que añadir bastante información a los paquetes enviados.

Debido a que estos paquetes tienen un tamaño máximo, cuanta más información se añada al protocolo para su gestión, menos información que proviene de la aplicación podrá contener ese paquete (el segmento TCP tiene una sobrecarga de 20 bytes en cada segmento, mientras que UDP solo añade 8 bytes). Por eso, cuando es más importante la velocidad que la fiabilidad, se utiliza UDP. En cambio, TCP asegura la recepción en destino de la información para transmitir.

Ya que tanto TCP como UDP circulan por la misma red, en muchos casos ocurre que el aumento del tráfico UDP daña el correcto funcionamiento de las aplicaciones TCP. Por defecto, TCP pasa a un segundo lugar para dejar a los datos en tiempo real usar la mayor parte del ancho de banda. El problema es que ambos son importantes para la mayor parte de las aplicaciones, por lo que encontrar el equilibrio entre ambos es crucial.

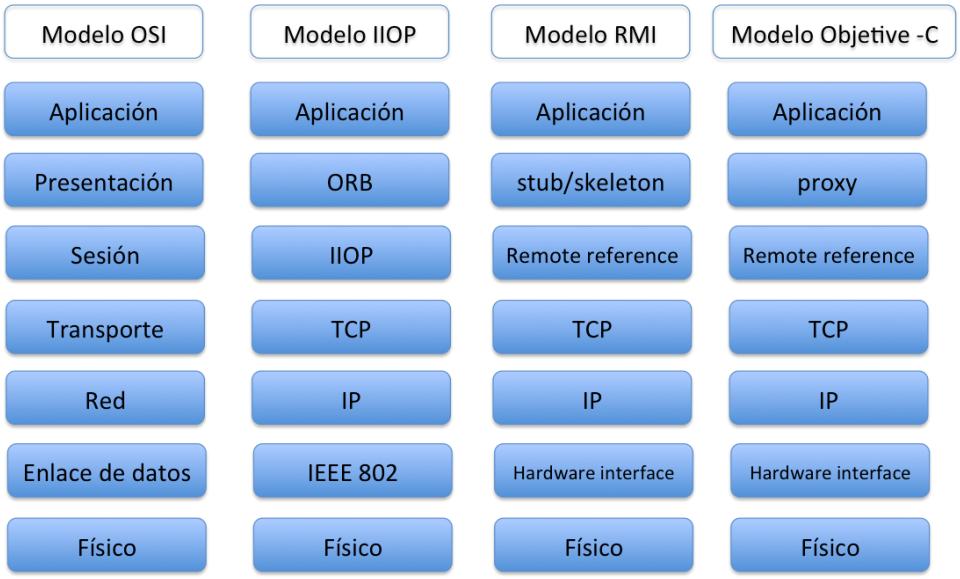


Figura 2.21: Capas del modelo OSI de objetos CORBA, RMI y Objetive-C

Diversas combinaciones de protocolos en cada capa permiten una distribución segura de información, o un envío de paquetes de mayor tamaño, los protocolos que buscan la ruta más corta pueden ser la mejor opción en casos donde el envío tiene que viajar desde dos puntos muy lejanos. En particular la implementación de los objetos CORBA, RMI y Objetive-C que se mencionaron con anterioridad utilizan cierta combinación de protocolos (ver figura 2.21).

Capítulo 3

# Análisis

Los diversos desarrollos orientados al manejo de paredes de vídeo se pueden clasificar en render paralelo, memoria compartida y tarjetas en cascada. Estos desarrollos requieren el uso de tecnologías encontradas dentro de las áreas de visualización, sistemas distribuidos, interacción humano-computadora y redes. Es por eso que en este capítulo se hará un análisis de la interacción de todos estos desarrollos en las diferentes áreas.

En el control de las paredes de vídeo hay aspectos que son muy importantes como la interfaz de usuario; estos aspectos varían con respecto al modelo lógico empleado, además los dispositivos de entrada que pueden ser utilizados son múltiples. Existen reglas de diseño de HCI que permiten realizar métricas con respecto a la funcionalidad final.

En nuestro modelo no se consideran características específicas dentro de la interfaz de usuario, sino que se quiere llegar a una generalidad que pueda extenderse mediante la adición de dispositivos de entrada (sin alterar la funcionalidad).

Otro aspecto importante es el envío de mensajes que requiere medios de transporte estratégicos para cada tipo de dato, como se mencionó en el patrón de diseño Strategy del capítulo anterior. En general todos los patrones de diseño mencionados con anterioridad explican la funcionalidad que será implementada dentro del Framework DVO.

Todos estos aspectos pueden ser obtenidos como estrategias que permitan el control de la pared de vídeo, ya que independientemente de lo explícitos o implícitos que estén en los sistemas actuales, forman parte de todas las soluciones.

A continuación se muestra la abstracción jerárquica del problema y el modelo conceptual del manejo de la pared de vídeo. También se muestra el diseño y funcionamiento de una aplicación implementada en el Cinveswall, con la finalidad de señalar los requerimientos de paredes de vídeo de este tipo.

## 3.1. Análisis del los modelos de control de paredes de vídeo

Revisemos el modelo conceptual (ver figura 3.1). Existe un usuario que interactúa con un servidor, el servidor controla a un grupo de nodos, el grupo de nodos despliega su salida en un grupo de pantallas. El servidor que controla a los nodos es el servidor de visualización y el grupo de nodos es el cluster de visualización. Los principales datos manejados en el servidor de visualización y en el cluster de visualización son los objetos de despliegue.

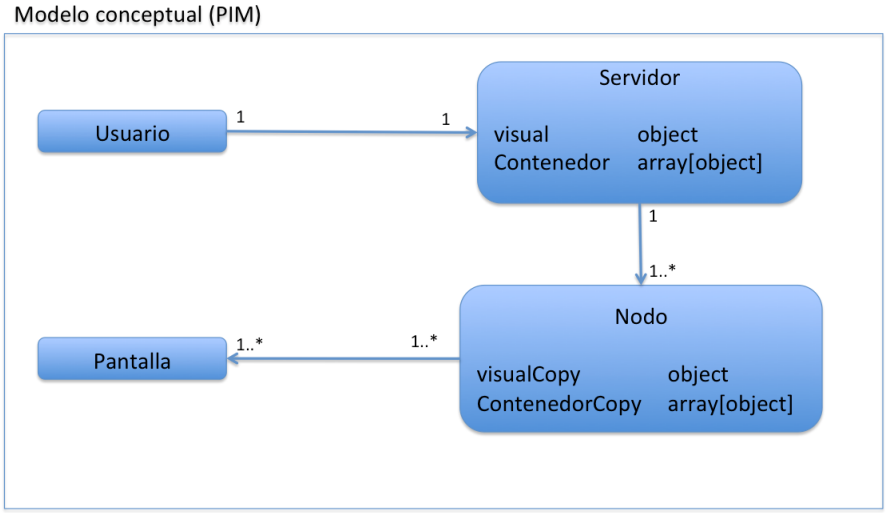


Figura 3.1: Modelo conceptual (PIM)

Podemos dividir las características generales de un manejador de paredes de vídeo separando aspectos de alto nivel de las funciones físicas. Además se deben separar las prioridades de funcionalidad y las restricciones que se tienen que en ocasiones involucran los componentes físicos, de modo que la única capa que podemos separar de todas, que se mantiene independiente de las restricciones y las metas de alto nivel son las funciones generalizadas; estas funciones representan el conjunto básico de requerimientos para el control de la pared de vídeo.

En este caso debemos notar la importancia de las funciones generalizadas, ya que son el núcleo que permite el funcionamiento de las paredes de vídeo independientemente de la aplicación final. Dependiendo del enfoque tecnológico aplicado estas funciones son más o menos detalladas.

Empleamos una abstracción jerárquica del problema para proporcionar el modelo general de los requerimientos de un sistema manejador de paredes de vídeo (ver figura 3.2).

Las funciones generalizadas se pueden dividir en las siguientes: distribución de datos, sincronización, manejo de eventos y distribución de despliegue. Las cuales deben funcionar sobre el hardware disponible. En las siguientes secciones se presentan estas funciones dentro de los sistemas mencionados en el estado del arte.

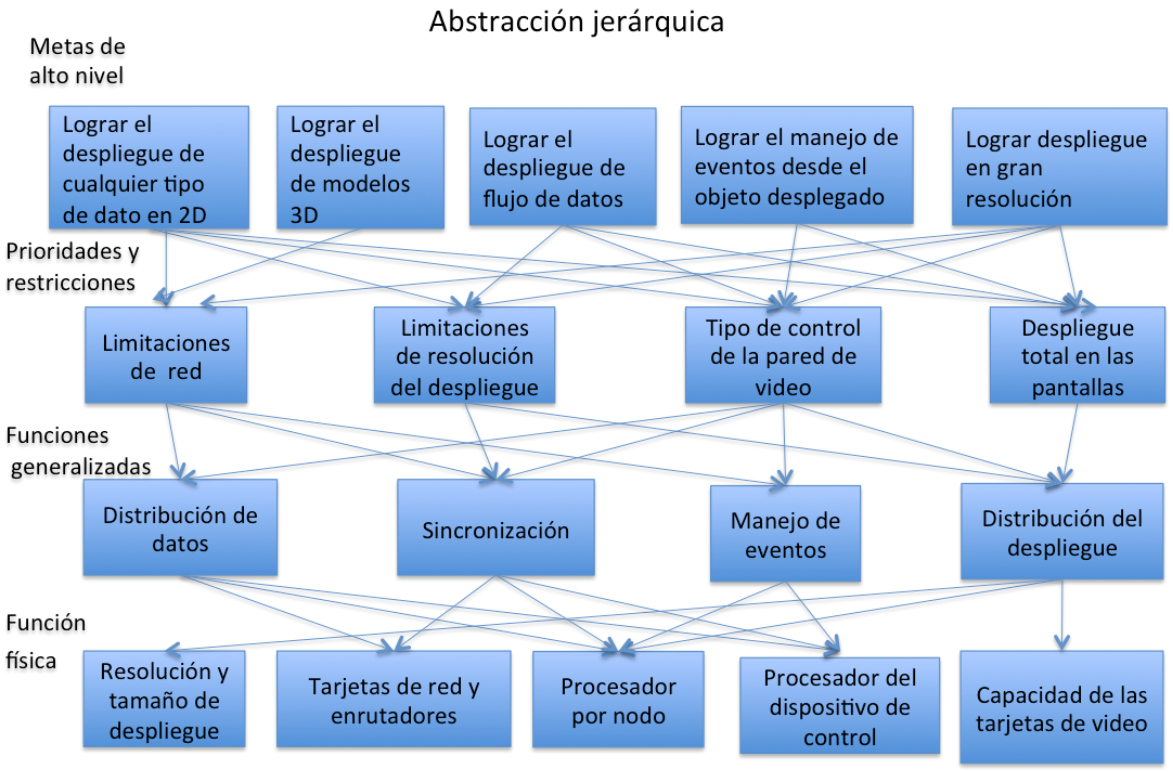


Figura 3.2: Abstracción jerárquica del problema del manejo de la pared de vídeo, muestra la clara separación existente entre los componentes hardware y las características proporcionadas por la pared de vídeo.

### 3.1.1. Análisis de modelos propuestos en el estado del arte

Unos de los principales enfoques empleados para el manejo de paredes de vídeo lo dan los proyectos que emplean memoria compartida (ver figura 3.3). La memoria compartida permite almacenar información generada desde interfaces remotas y permitir su acceso desde los nodos encargados del despliegue. Esta memoria se emplea como un contenedor de salidas gráficas de diversas aplicaciones. De esta manera es posible desplegar cualquier tipo de aplicación en la pared de vídeo.

Mantener la memoria compartida proporciona integridad en la información desplegada, además facilita su actualización. El acceso de los nodos es directo y esto puede ocasionar cuellos de botella, cuando el número de nodos es elevado. La cantidad de memoria que requiere el contenedor es elevada y no siempre es ocupada por completo.

El contenido de la memoria compartida en este caso es un conjunto de matrices de despliegue. Cada matriz de despliegue es obtenida desde las interfaces remotas; el tamaño ocupado en memoria por cada matriz esta definido por la resolución y profundidad de color de la tarjeta gráfica de la interfaz remota.

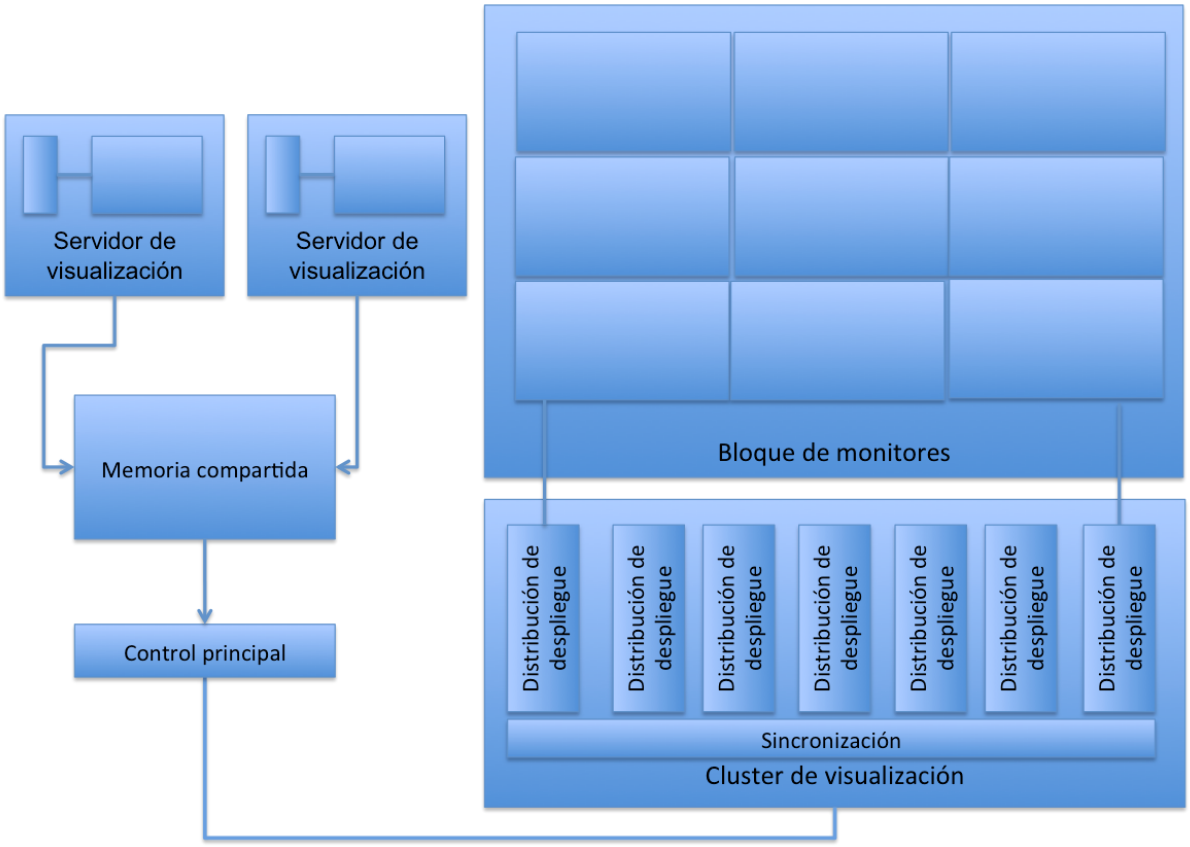


Figura 3.3: Modelo de memoria compartida (Distribución de despliegue)

### Distribución de despliegue

En particular existe una modalidad de memoria compartida que toma el contenido del frame buffer de la tarjeta de vídeo de la interfaz gráfica remota (protocolo VNC). Cuando se toma la información directamente del frame buffer se obtiene una matriz de pixeles lista para desplegar. En este caso se requiere de la *distribución de despliegue*, que es la encargada de dividir la matriz de pixeles y asignar porciones de la matriz a los nodos. Esta distribución requiere conocer la resolución y posición de cada pantalla.

Ejemplificamos este proceso mediante un algoritmo de distribución de despliegue usando máscaras, para dividir una imagen de alta resolución y gran tamaño, distribuida en el despliegue formado por un número de pantallas, donde cada pantalla tiene una resolución, y la suma de las resoluciones es la resolución total de la pared de vídeo.

Este algoritmo obtiene la distribución de la matriz de imagen adecuada para cada pantalla mediante el recorte usando una máscara. Cada máscara se asigna por posición de pantalla y se recorta en la matriz de la imagen original para obtener la parte desplegable para cada pantalla.

Inicialmente se debe realizar un emparejamiento de tamaños de la matriz máscara y

el tamaño de la imagen original. Las máscaras de posicionamiento son asignadas desde la configuración inicial. El proceso de recorte se realiza en cada nodo para distribuir el trabajo.

**Algorithm 1** Distribución de imágenes usando máscaras

```
Require: Matriz máscara de la resolución total de la pared de vídeo rtP
Require: Imagen original del tamaño de la matriz máscara
Ensure: x= alto de la matriz máscara
Ensure: y= ancho de la matriz máscara
Ensure: MR=matriz resultado
Ensure: MM=matriz máscara
Ensure: MO=matriz original
Ensure: MF=matriz final
  for i=1 to x do
    for j=1 to y do
      MR[x][y]= MM[x][y] and MO[x][y]
    end for
  end for
  MF=BorraZeros(MR)
```

*MF* es la matriz final, que será desplegada por cada nodo. El tamaño de la matriz MM esta definido por las coordenadas $(x, y)$ obtenidas de la resolución total de la pared de vídeo $rtP$. Después del proceso de emparejamiento el tamaño de la matriz original de la imagen $MO$ se hace igual a $MM$ (ver figura 3.4). Cuando se necesite el uso de la pared de vídeo completa es necesario escalar el tamaño de la imagen original. El proceso de fracción se puede hacer en el cliente o en cada servidor. En este caso se propone realizar la división dentro de cada servidor a fin de distribuir el trabajo.

### Sincronización

La distribución del despliegue es fundamental dentro de las tecnologías de control de paredes de vídeo basadas en memoria compartida, es en este enfoque donde se trabaja principalmente con matrices de pixeles obtenidas desde diversos dispositivos de entrada. Otra importante tarea que se debe considerar dentro de la distribución de despliegue es el modo de sincronización de las pantallas. La sincronización en este tipo de tecnología esta enfocado en *el despliegue al mismo tiempo en todas las pantallas*, ya que una de las características de las paredes de vídeo es permitir su uso como si fuera una sola pantalla. Es por eso que cuando empleamos el modelo de frame buffer compartido (ver figura 3.5), la *sincronización* principal la encontramos en los bloques de distribución de despliegue.

Cuando se trabaja con múltiples máquinas al mismo tiempo se generan diversos problemas de sincronización, ya que todas las máquinas muestran una diferencia de tiempo en su reloj [1], por ello existen servidores de tiempo encargados de proporcionar el mismo

[1]El reloj de cada máquina emplea la oscilación de un cristal para mantener su tiempo y los cristales

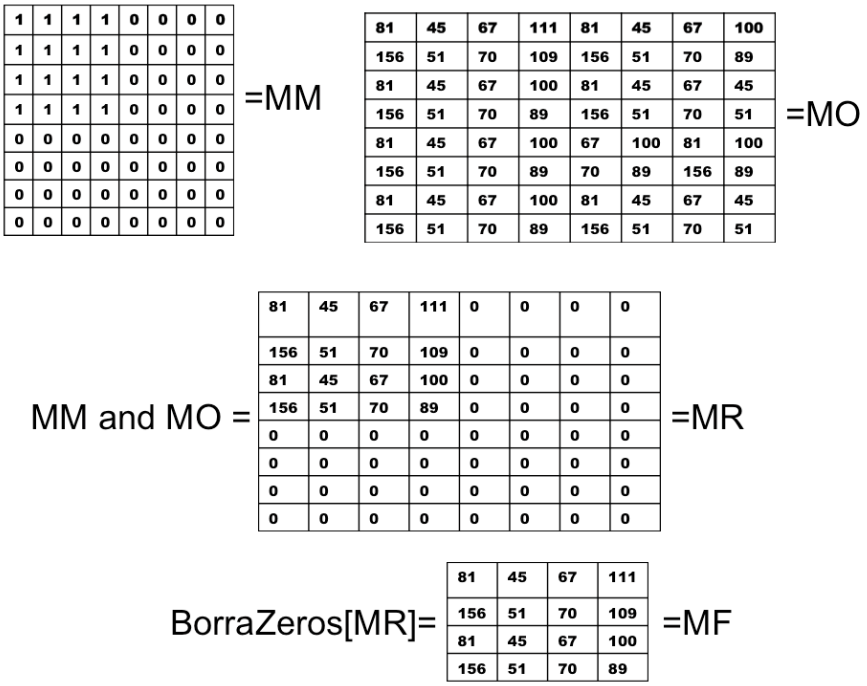


Figura 3.4: Ejemplo del algoritmo de distribución basado en máscaras

tiempo a todos los nodos.

Es posible sincronizar el despliegue empleando mensajes de tiempo entre el control y los nodos encargados del despliegue. Para esto se mide el tiempo que ocupa cada nodo en desplegar (tn), además debemos conocer el tiempo que se requiere para enviar la señal de despliegue del control a cada nodo (tcn); esta información debe ser almacenada en una lista ya que el tiempo para cada nodo es diferente. Cuando tenemos los tiempos (tn,tcn) calculamos el retraso de despliegue para cada nodo. El control es ahora quien debe indicarle a cada nodo el tiempo que debe esperar para realizar su despliegue. Por ejemplo si un nodo tarda 5 segundos y otro tarda 10 segundos, el control debe indicarle al nodo que tarda 5 segundos que espere 5 segundos más. En las pruebas que se realizaron sobre el CinvesWall se encontró que dependiendo de la resolución y la compresión del archivo existían mayores problemas en el despliegue. Además en comparativas realizadas usando notificaciones e invocaciones remotas, se obtuvo que las notificaciones mantenían un tiempo más constante en los nodo que las invocaciones remotas. A nivel físico para esta tecnología, se sabe que el Broadcast es mejor opción que el peer to peer, pero eso es dependiente del hardware disponible.

no oscilan a la misma velocidad.

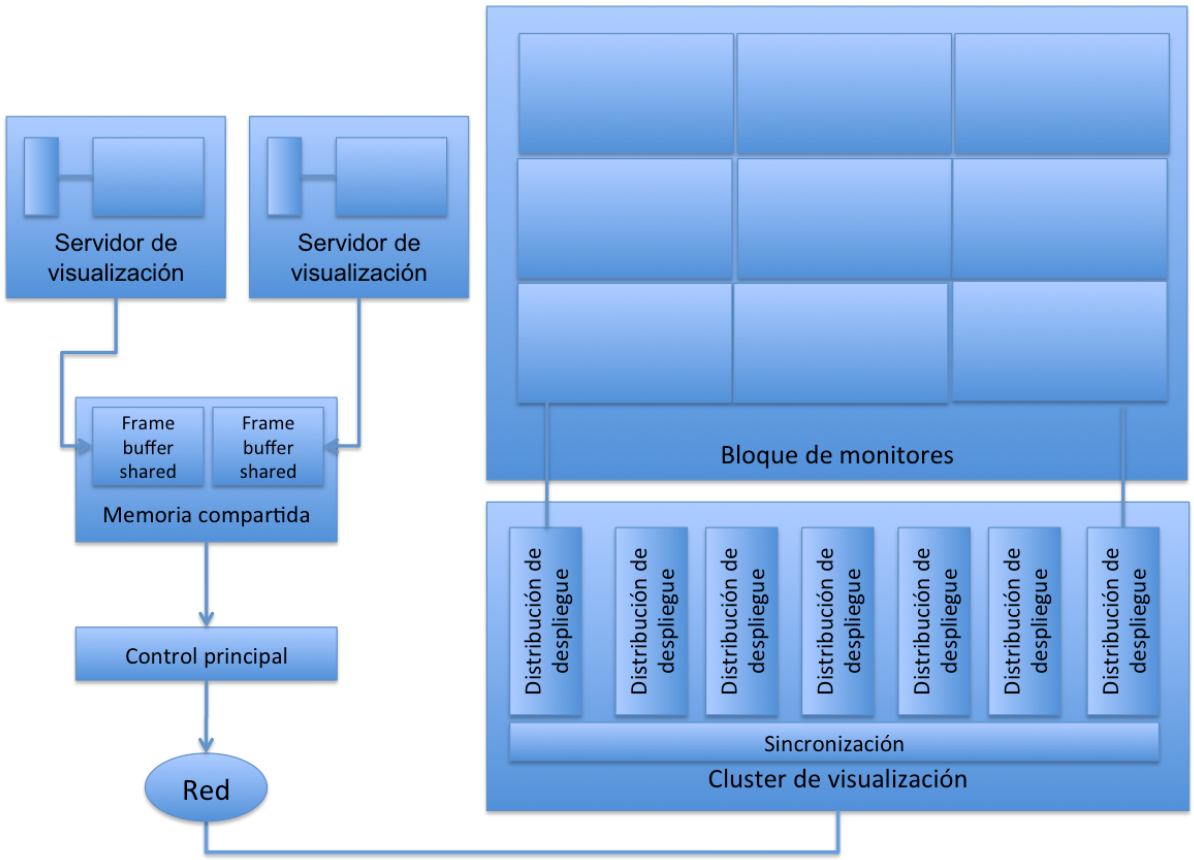


Figura 3.5: Modelo con frame buffer shared

### Distribución de datos

El render paralelo es otra aproximación importante en el control de paredes de vídeo, ya que permite dividir la carga de trabajo en la realización del render de un modelo 3D. Como se mencionó en el estado del arte la transformación de un modelo 3D a una matriz 2D que puede desplegarse en pantalla requiere diversos procesos. Dentro de cada proceso se puede hacer una distribución de datos para lograr mantener el trabajo en todos los nodos del cluster de visualización. La mayoría de los trabajos realizados con este enfoque se apoya en la biblioteca OpenGL.

Para obtener la matriz desplegable de un objeto 3D debemos elegir una perspectiva desde donde se mire el objeto. Existen diferentes técnicas que nos permiten obtener estas perspectivas. Cuando se realiza una transformación sobre el objeto la información de nuestra perspectiva debe cambiar. Al extender esto en una pared de vídeo tenemos que el render paralelo obtiene ventajas y desventajas dependiendo de la etapa en la que se realice la distribución. Sobre todo en las transformaciones realizadas después del despliegue inicial. Es decir aunque generemos el render de una perspectiva del objeto, cuando

se realice un cambio necesitamos realizar el render otra vez. Es por eso que los modelos 3D de OpenGL trabajan empleando un contexto, donde todas las características del objeto son almacenadas y conservadas durante todo el tiempo de despliegue; OpenGL solo proporciona las primitivas básicas de dibujo, la interacción se obtiene de la biblioteca GLUT.

Es por esto que cuando se trabaja con render paralelo se selecciona un conjunto de nodos de despliegue (con sus respectivas pantallas), que no cambiarán su posición en tiempo de ejecución manteniendo el despliegue en una posición específica sobre un grupo de pantallas.

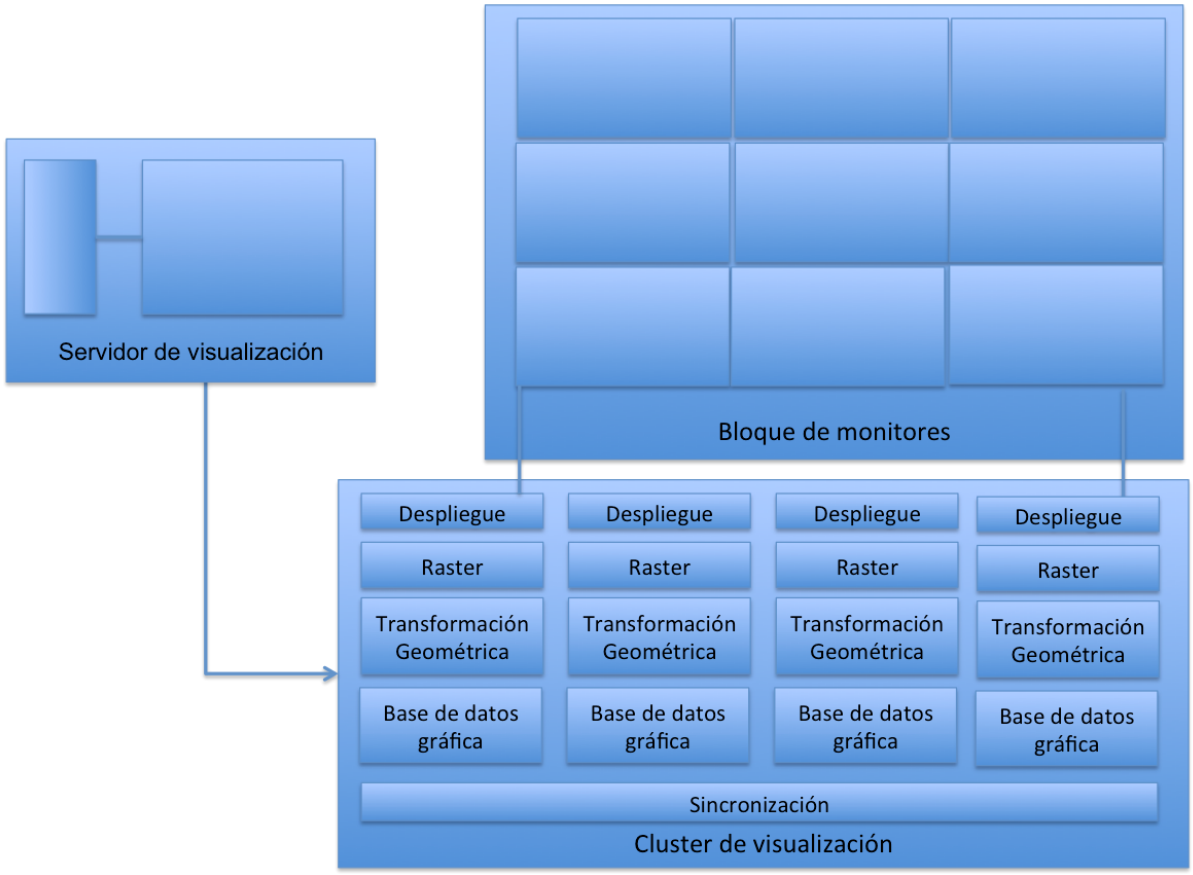


Figura 3.6: Modelo con render paralelo

El proceso de render se puede realizar dentro de una tarjeta gráfica o con la unión de varias. Por ejemplo en la imagen (ver figura 3.6), se puede ver como el control esta dentro de una máquina que envía la información a un grupo de nodos que realizan el render y están conectados a las pantallas de despliegue. En este caso se debe realizar la distribución de los datos a nivel del control, el tipo de distribución se puede elegir con respecto a la etapa del render como se mencionó con anterioridad.

Si extendemos el diagrama del render paralelo podemos ver el nivel donde se realiza

la sincronización para cada nodo. Hay que tomar en cuenta además que los algoritmos de distribución en estas etapas son complejos, ya que lo que se divide es un modelo en 3D (vértices con coordenadas x,y,z) lo cual incrementa la dificultad en el recorte. Esta dificultad se puede comparar con la generada al dibujar una línea en diagonal; la forma rectangular de los pixeles nos obliga a elegir entre pixeles cercanos y duplicados de manera que es posible simular la línea. En el caso de un modelo con coordenadas tridimensionales se emplean en ocasiones mallas de polígonos que permitan separar los objetos y formar grupos de vértices cercanos que pueden ser desplegados en diferentes máquinas.

Existen estrategias que permiten la distribución de datos (replicación, fragmentación, migración, etc.) en cada caso se generan problemas y beneficios dependiendo de la etapa en la que se encuentre el render.

En el caso de render paralelo mantener replicada la información inicial permite que cuando se genere una transformación, no se necesite obtener de nuevo los datos iniciales. Se usa más memoria pero se ahorra tiempo y se disminuye el número de mensajes. En el caso de fragmentar la información por ejemplo a nivel de geometría, permite usar menos memoria (memoria de tarjeta de vídeo) pero emplea más mensajes y eso se puede reflejar en un retraso.

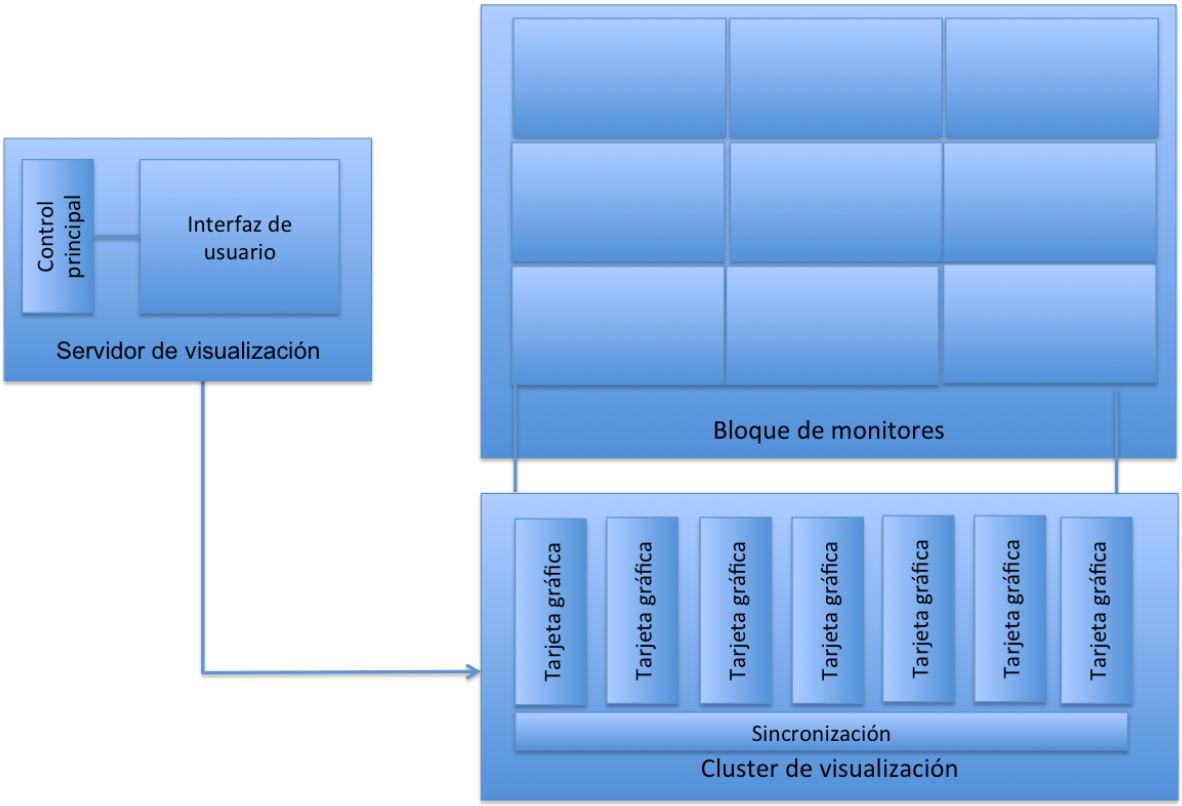


Figura 3.7: Modelo usando tarjetas en cascada

El último enfoque basado en el uso de tarjetas en cascada (ver figura 3.7), funciona de manera semiautomática ya que existen interfaces gráficas que permiten la configuración de la pared de video. Las tarjetas gráficas de última generación trabajan con múltiples (cientos) GPUs con memoria individual, que muestran el funcionamiento ideal del render paralelo. La limitante en el uso exclusivo de las tarjetas gráficas es que aunque es posible incrementar el tamaño de la pared de video, la resolución esta limitada a la de la tarjeta.

### 3.1.2. Manejo de eventos

El manejo de eventos como se realiza en manejadores de ventanas de escritorios estándar muestra diversas variantes, dos de las principales son basadas en tablas como en el caso de QT y mediante delegados como es el caso de Cocoa. En el primero se mantiene una tabla con los eventos que pueden ocurrir y las acciones relacionadas, de manera que cuando un evento ocurre se realiza una búsqueda dentro de la tabla para realizar su correspondiente acción (la información obtenida se almacena en un objeto Qevent y se ejecutan sus métodos para realizar la acción). En el segundo caso se mantiene un grupo del cual se pueden heredar los métodos que realizan las diversas acciones consecuencias de un evento. En el caso de Cocoa se mantiene un NSResponder del cual se heredan los métodos específicos para responder a eventos de teclado y de ratón (la información obtenida se almacena en un objeto NSEvent que hereda sus métodos del NSResponder).

En el caso del manejo de eventos de paredes de vídeo se extiende la idea anterior para poder obtener el manejo de eventos remoto (ver figura 3.8). Los proyectos con memoria compartida realiza el manejo de eventos dentro de la interfaz de usuario remota que es la que proporciona la salida gráfica a la pared de vídeo. El reconocimiento del evento se realiza de manera similar al caso de manejadores de ventanas mono usuario. La diferencia inicia al generar un mensaje con la información de lo sucedido, y realizar una duplicación del mensaje del evento para ser enviado a los nodos, quienes analizarán el mensaje y realizarán la acción que sea necesaria.

La facilidad de adición de dispositivos de entrada dentro de las paredes de vídeo es muy importante, ya que se contempla la integración de tecnologías de reconocimiento de movimientos corporales; esto se realiza mediante cámaras que permiten identificar la posición del usuario (empleando algoritmos de seguimiento de objetos). Por ejemplo, el Kinect que es un dispositivo de entrada con sensor de movimiento; emplea un sensor de profundidad infrarrojo combinado con un sensor monocromático y un componente en hardware que realiza el seguimiento de los objetos.

Otra tecnología que se ha usado en paneles interactivos similares a las paredes de vídeo es touch screen ya que permite accesibilidad de uso a los usuarios. La ventaja de este tipo de dispositivos es que cuentan con APIs que facilitan su integración en ambientes como las paredes de vídeo, en cuanto a sus beneficios en HCI todavía se encuentra en investigación.

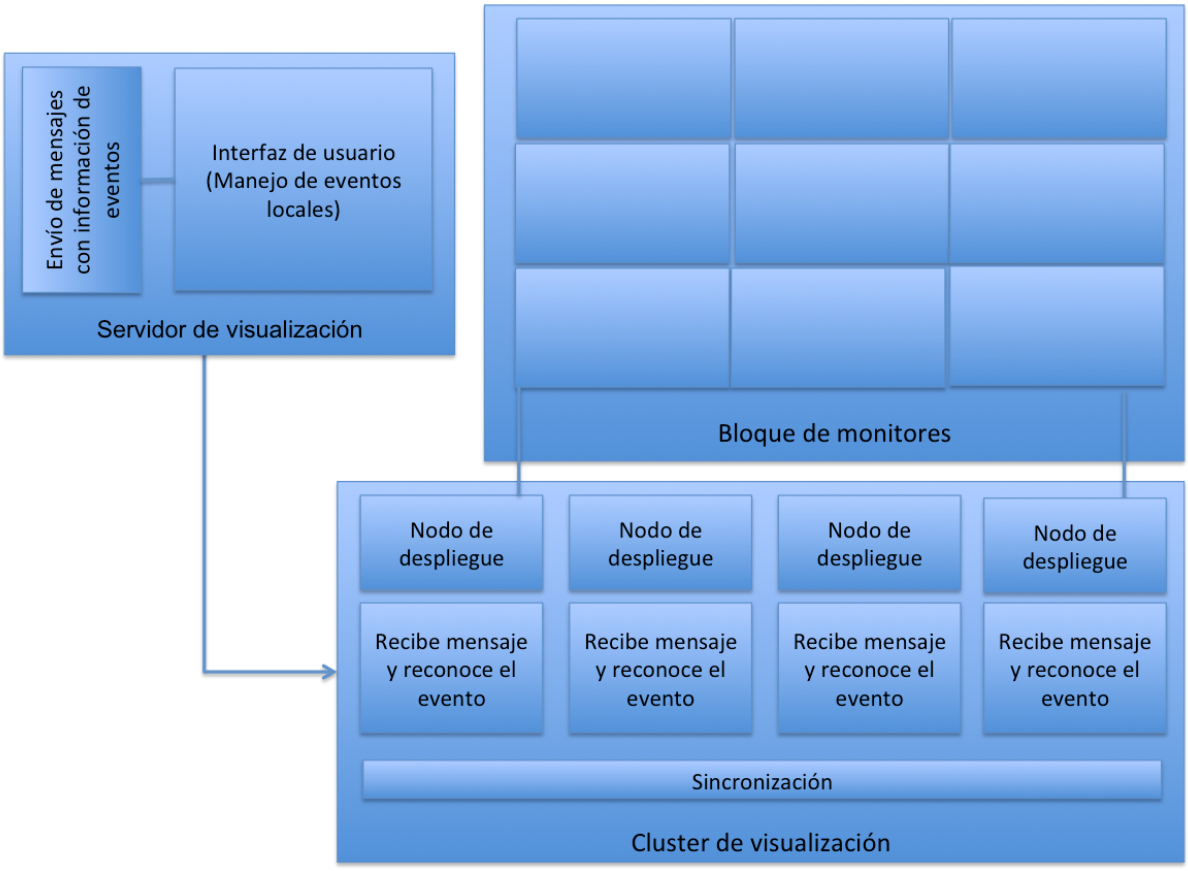


Figura 3.8: Modelo de manejo de eventos en una pared de vídeo

### 3.1.3. Conclusión del análisis

De los enfoques mencionados con anterioridad usar el frame buffer compartido permite desplegar cualquier tipo de aplicación dentro de la pared de vídeo, pero implica un uso excesivo de la red y la memoria del contenedor.

Para compensarlo se creo el flujo de datos en el cual se envía la información a desplegar almacenandola en una memoria cache de grandes dimensiones. El uso intensivo de la red se ha solucionado mediante redes ópticas de alta velocidad que permiten el envío de hasta 10Gb de información. Esto lo convierte en una tecnología excluyente ya que la mayoría de los laboratorios no especializados solo cuentan con una red Ethernet.

Es por eso que proponemos una solución basada en el uso del frame buffer compartido distribuido, de modo que sea posible disminuir el tamaño del paquete y pueda transmitirse en una red común (ver figura 3.9). Para evitar los cuellos de botella podemos dirigir la información a un cluster de distribución que se comunique con el cluster de despliegue.

Por ejemplo cuando se trata de visualización científica, es posible distribuir una base de datos con la información a desplegar, de modo que cada nodo puede generar su despliegue

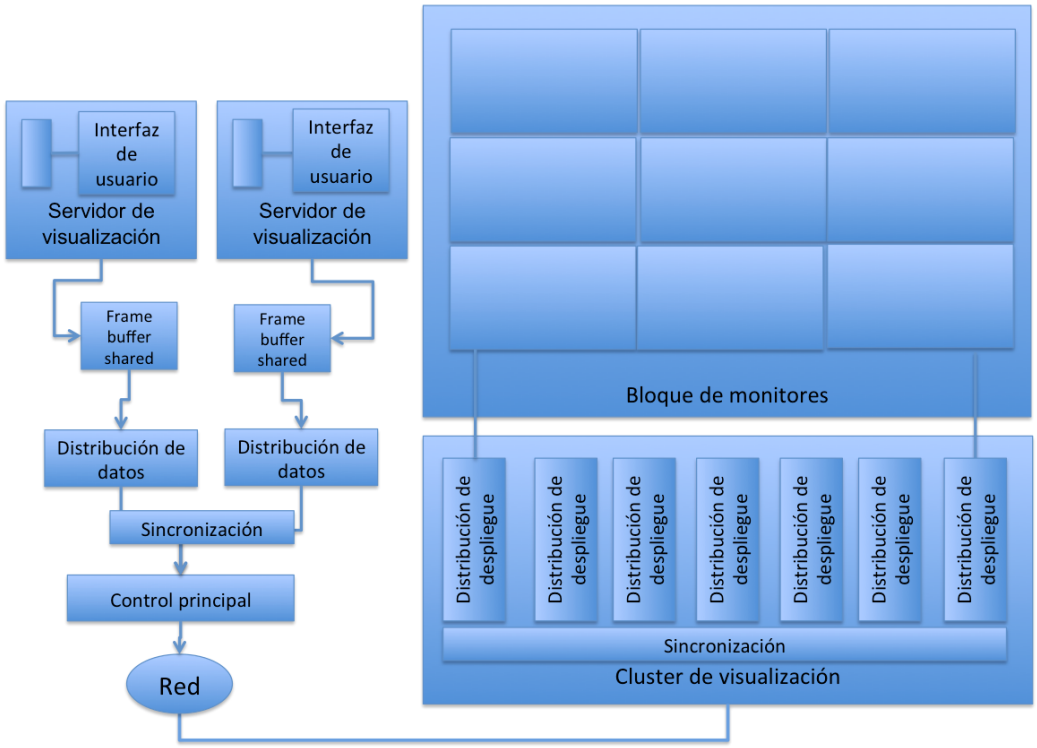


Figura 3.9: Modelo con frame buffer shared con distribución de datos sincronizado

sin sobre cargar la red. Similar al render paralelo permite el despliegue de un modelo 3D en un conjunto de máquinas distribuyendo el procesamiento en cada una de ellas.

Finalmente se propone el uso de un modelo DVO, que se base en las estrategias básicas para optimizar las funciones generalizadas. De esta manera es posible aprovechar las diferentes ventajas mencionadas en los modelos anteriores (ver figura 3.10). En este modelo la sincronización se realiza del lado de los nodos pero implica recibir la información del control principal. El control principal se encarga de proporcionar las estrategias para la distribución de los datos, esto mediante el protocolo DVO, donde se indican los métodos remotos que estarán habilitados para el control de la pared de vídeo. Podemos dividir estos métodos en los encargados del manejo de eventos y los encargados de los datos de despliegue.

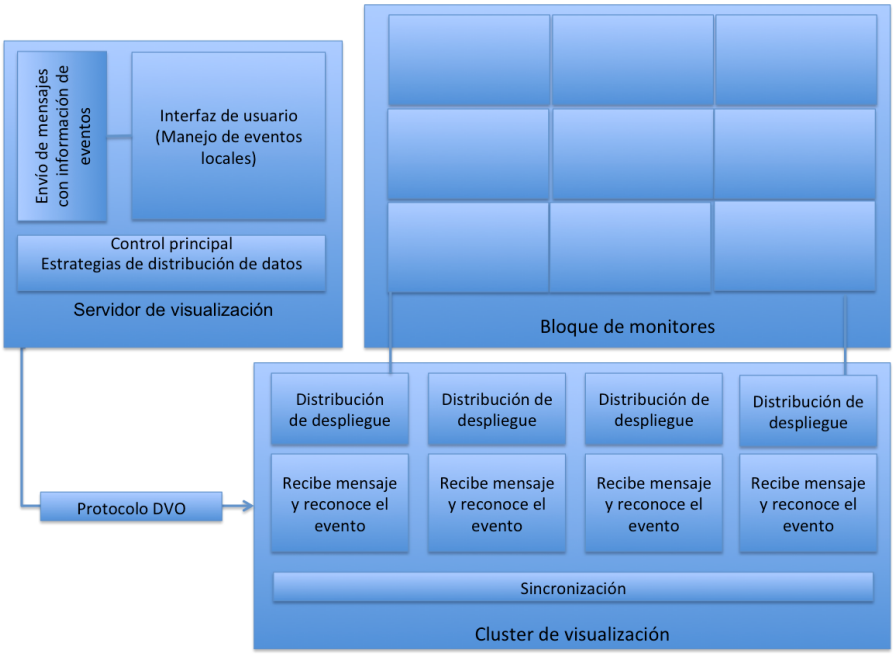


Figura 3.10: Modelo DVO

Capítulo 4

# Modelo DVO

En este capítulo se presenta un modelo para la definición de los objetos distribuidos visuales (DVO). Este modelo proporciona los lineamientos generales para la creación de aplicaciones en ambientes de paredes de vídeo. La descripción de este modelo muestra las etapas donde se encuentran las funciones generales mostradas en el capítulo anterior. Esta descripción permite desarrollar la idea del manejador de la pared de vídeo independientemente de la aplicación final. En la primera sección se explica el modelo DVO definido en capas. La sección siguiente proporciona una formalización de los conceptos y propiedades del DVO.

El modelo DVO está definido en capas (ver figura 4.1), las cuales funcionan de modo independiente. Las capas se relacionan formando un bloque, este bloque se encuentra entre el sistema operativo y las aplicaciones. Este modelo se ha definido con la finalidad de proporcionar la base de lo que a futuro será un manejador de ventanas distribuido. Durante los capítulos anteriores se abordaron los temas necesarios para discutir este modelo. La descripción que a continuación se presenta muestra una vista general del modelo y la descripción de cada capa que lo conforma.


Figura 4.1: Posición de la capa de objetos distribuidos visuales (DVO)

En el capítulo anterior se mostraron los modelos que funcionan sobre las paredes de

vídeo, sus ventajas y desventajas solventadas en la mayoría de los casos por el hardware utilizado. En este modelo por el contrario se busca obtener una solución que permita emplear las estrategias de las funcionalidades generalizadas, para mantener el concepto independiente del hardware.

El modelo propuesto se basa en el paradigma orientado a objetos. Específicamente en el uso de objetos distribuidos que permiten la comunicación remota entre el servidor y los nodos. Además se puede tomar ventajas de la distribución de su estado (distribución de datos), y se puede controlar el tiempo de envío de mensaje (para lograr sincronización). Dentro de los conceptos obtenidos de un manejador de ventajas de escritorio estándar obtenemos un componente capaz de desplegar cualquier tipo de información y además recibir eventos de usuario que en este trabajo hemos denominado objeto visual. De tal manera que el modelo DVO, obtiene su funcionalidad mediante la unión de objetos distribuidos y objetos visuales. Para que estos objetos trabajen es necesario permitir funcionalidades básicas (comunicación, despliegue y manejo de eventos).

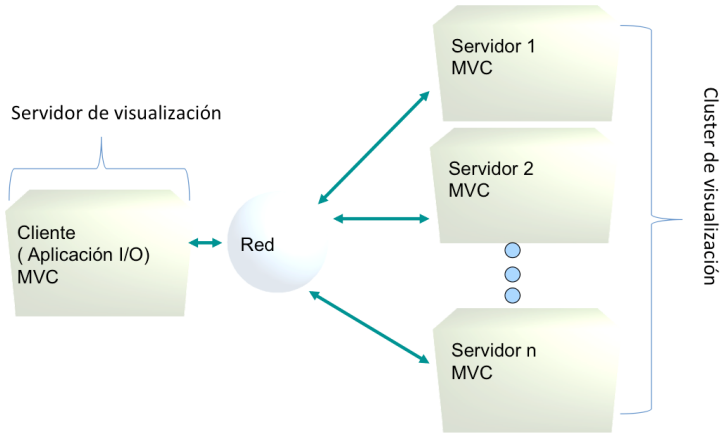


Figura 4.2: Modelo DVO(MVC)

El modelo esta definido en capas semejante al patrón MVC (ver figura 4.2). El patrón de diseño MVC permite encapsular los datos de la aplicación en el modelo, manteniendo el despliegue y la edición de datos en la vista, y el control que permitirá el trabajo de los dos. Como se menciono con anterioridad el manejo lógico del tiled display se realiza mediante el modelo cliente servidor. En esta modificación el control del cliente se comunica con el control del servidor de modo que es posible emplear el modelo del cliente para desplegar en las vistas de los servidores.

El servidor de visualización es el encargado de recibir los eventos del usuario y de enviar la información al cluster de visualización. De forma que el procesamiento necesario para el despliegue en alta resolución se realice directamente sobre el cluster de visualización.

El servidor de visualización genera la distribución de datos y se despliega una versión mínima a escala de los objetos que se desplegarán en el cluster de visualización. Por un

lado tenemos las características de distribución que se requieren en este modelo (basadas en comunicación mediante mensajes y el acceso remoto). Por otro lado los aspectos visuales necesarios como el manejo de eventos y despliegue de cualquier tipo de información. La unión de estas características representa el conjunto mínimo que conformará la funcionalidad del DVO. Estas características se pueden definir en capas independientes. Estas capas constituyen la definición del modelo DVO. En la siguiente sección se describen cada una de las capas del modelo. Además se muestran algunas de las clases básicas del API DVO, algunas de ellas surgen directamente del modelo, otras aparecen por necesidades propias de cada capa. A continuación mostraremos las clases que constituyen cada capa y las descripciones de sus métodos.

| API de inicialización DVO | | | |
|---|---|---|---|
| Nombre | Parametros | Descripción | Error |
| DVOInicializarServidor | NetConfigWall<br>DisConfigWall | Esta función toma los valores de configuración de red del archivo NetConfigWall y de configuración de despliegue DisConfigWall. Una vez alamacenada la configuración es posible iniciar la conexión con los nodos. | 101.- No se encontro el archivo NetconfigWall<br>102.- No se encontro el archivp DisConfigWall |
| DVOInicializarNodos | DisConfig | Esta función toma la información de posición local de cada nodo almacenada en el archivo DisConfig, donde además se almacena el tamaño total del despliegue y la resolución de cada pantalla. | 201.- No se encontro el archivo DisConfig.<br>202.- El archivo no tiene el formato adecuado. |
| DVOEstablecerConexión | | Esta función se realiza después de las inicializaciones y es la encargada de habilitar las conexiones que sean necesarias entre el servidor y los nodos. | 301.- No se pudo establecer la conexión.<br>302.- Puertos ocupados. |
| DVOAgregarEvento | Tecla1 y Tecla2 | Esta función permite agregar combinaciones de teclas en la tabla de eventos. | 301.-Las teclas requeridas han sido asignadas previamente. |

Figura 4.3: API de inicialización

El API DVO permite el manejo de paredes de vídeo empleando el modelo DVO (ver figura 4.3). Esta API esta pensada para permitir utilizar la funcionalidad de diferentes modos de comunicación empleados en los mensajes remotos. Se trata de una API que permite la creación de aplicación de diversos tipos mediante el uso de las clases encargadas de la visualización. Esta API proporciona el manejo de diversos tipos de visualizaciones, así como la distribución de los datos de origen. La siguiente sección contiene un listado detallado de acciones que se pueden ejecutar a través de la misma. Las configuraciones iniciales serán obtenidas en la mayoría de los casos por archivos de configuración. Esto debido a que la arquitectura más común suele ser cliente-servidor. Por lo que la configuración de puertos e IP's de los nodos despliegue es siempre constante, así como la disposición del arreglo de pantallas y los tamaños de cada una.

## 4.1. Capas del modelo DVO

En el modelo en capas (ver figura 4.4), cada capa cumple con un conjunto de funciones específicas de forma que pueden funcionar de manera independiente. Las capas de bajo nivel son las que tienen contacto directo con el sistema operativo. La superior se relaciona directamente con la aplicación final.

La capa de comunicación proporciona las funcionalidades básicas para la transmisión de mensajes entre los nodos. La capa de control mantiene comunicación con las capas inferiores y superiores, ya que es la encargada de distribuir las funcionalidades a cada capa. Dentro de las capas que encontramos en el servidor podemos eliminar temporalmente al manejo de eventos, de modo que la interfaz gráfica de usuario esta únicamente del lado de cliente.

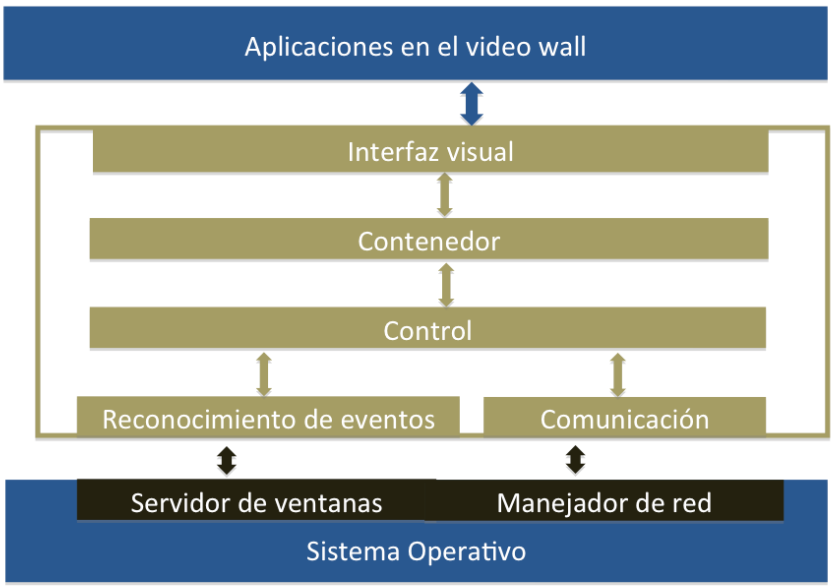


Figura 4.4: Componentes principales de la capa de objetos distribuidos visuales (DVO)

### 4.1.1. Capa de la interfaz visual

Uno de los aspectos principales que permiten la funcionalidad del sistema involucra la definición del objeto encargado del despliegue. Además requiere manejar eventos ocurridos dentro de su área. Este objeto se llama objeto visual y se define a continuación:

*Objeto Visual (VO). Un objeto visual es un área rectangular que permite el despliegue de visualizaciones y el manejo de eventos de usuario ocurridos dentro de esa área (ver figura 4.5).*

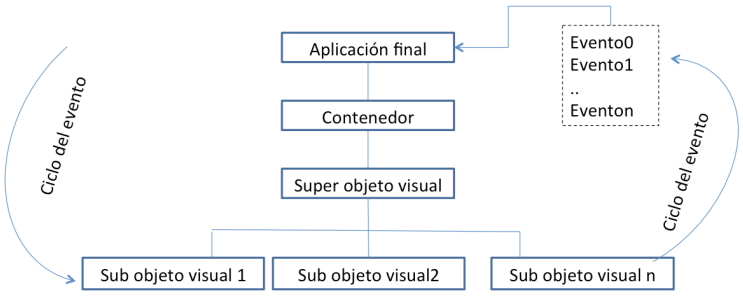


Figura 4.5: Modelo del objeto visual

Es posible definir al menos tres estados que un objeto visual puede tener. En el estado inicial (Start) está habilitado para recibir el dato inicial que va a desplegar. Cuando el dato inicial se visualiza el estado es Desplegado (Display). Del estado desplegado se cambia al estado de espera (Wait), este estado permanece hasta que un evento de usuario realice alguna de las dos opciones posibles (ver figura 4.6). La primera opción es agregar un nuevo objeto visual dentro del objeto o fuera de él. La segunda es actualizar el objeto visual cuando un evento ha ocurrido.


Figura 4.6: Estados del objeto visual

Las funciones básicas definidas en la API de esta capa se muestran en la figura 4.7. Estas funciones permiten la interacción directa con los objetos visuales y obtienen las funcionalidades necesarias para su funcionamiento de las capas inferiores. Los tipos de objetos visuales mantienen un identificador numérico para reconocerlos (1) para imagen, (2) para vídeo y (3) para modelo 3D. Una vez creado el objeto visual no se podrá cambiar de tipo.

| API de la capa de interfaz visual | | | |
|---|---|---|---|
| Nombre | Parámetros | Descripción | Error |
| DVORota | Objeto visual de tipo modelo 3D, Coordenadas (X,Y,Z) | Esta función realiza la rotación del modelo 3D contenido en el objeto visual. | 111.- El objeto visual seleccionado no es de tipo modelo 3D. |
| DVOEscala | Objeto visual de tipo modelo 3D, valor (tipo entero) | Esta función realiza el escalamiento del modelo 3D contenido en el objeto visual. | 211.-El objeto visual seleccionado no es de tipo modelo 3D. |
| DVOTraslada | Objeto visual de tipo modelo 3D Coordenadas (X,Y,Z) | Esta función realiza la traslación del modelo 3D contenido en el objeto visual. | 311.-El objeto visual seleccionado no es de tipo modelo 3D. |
| DVOFiltra | Objeto visual de tipo imagen, Filtro(tipo entero) | Esta función permite aplicar un filtro definido en la lista de filtros, sobre la imagen contenida en el objeto visual. | 411.-El objeto visual seleccionado no es de tipo imagen. |
| DVODeten | Objeto visual de tipo vídeo | Esta función permite detener la reproducción del vídeo contenido en el objeto visual. | 511.-El objeto visual seleccionado no es de tipo vídeo. |
| DVOInicia | Objeto visual de tipo vídeo | Esta función permite iniciar la reproducción del vídeo contenido en el objeto visual. | 611.-El objeto visual seleccionado no es de tipo vídeo. |
| DVOPausa | Objeto visual de tipo vídeo | Esta función permite pausar la reproducción del vídeo contenido en el objeto visual. | 711.-El objeto visual seleccionado no es de tipo vídeo. |
| DVOAgranda | Cualquier tipo de objeto visual, Alto( tipo entero), Ancho (tipo entero) | Permite agrandar o encoger el objeto visual. | |
| DVOPosiciona | Cualquier tipo de objeto visual, Coordenadas (X,Y) | Permite cambiar la posición del objeto visual. | |

Figura 4.7: API de la capa de interfaz visual

### Clases de la capa de la interfaz visual

Esta capa es la encargada del despliegue final de las visualizaciones y de la interacción con el usuario. En el capítulo anterior se mostró el por que seria necesario el uso de una clase abstracta para el objeto principal de esta capa y de la siguiente. Ahora definimos esta clase y explicamos el funcionamiento de sus métodos.

Clase Visual.- Esta clase es la encargada de mantener una visualización dentro de un área rectangular, además permite el reconocimiento de evento mediante cambios de estados. Estos cambios son recibidos por observadores que se encargan de obtener la información completa del evento desde la cola de eventos del sistema operativo.
Métodos de la clase

- Draw Visualiza el contenido del dato en la dirección (id), en la posición (position) del tamaño (Size).
- Delete Elimina la visualización contenida en el objeto visual.

Clase Imagen.- Esta clase es una especificación de la clase visual. Esta definido para visualizar información de tipo imagen.
Métodos de la clase

- Draw Visualiza la imagen contenida en la información obtenida desde la creación del objeto.
- Resize Modifica el tamaño de la imagen.
- Aplica Realiza la aplicación de algún tipo de filtro sobre la imagen contenida.
- Reposiciona Realiza el cambio de posición de la imagen.

Clase Vídeo.- Esta clase es una especificación de la clase visual. Esta definido para visualizar información de tipo vídeo.
Métodos de la clase

- Draw Visualiza el vídeo y lo muestra detenido en la creación del objeto.
- Resize Modifica el tamaño del vídeo.
- Reposición Realiza el cambio de posición del vídeo.
- Stop Para la reproducción del vídeo.
- Play Inicia la reproducción del vídeo.
- Adelanta Permite adelantar el vídeo hasta el fin del archivo.
- Retrasa Permite regresar la reproducción del vídeo hasta el inicio del archivo.

Clase Modelo.- Esta clase es una especificación de la clase visual. Esta definido para visualizar información de tipo modelo en 3D.
Métodos de la clase

- Draw Visualiza los puntos de las coordenadas del modelo.
- Transladar Translada el modelo en los valores dados para x,y,z.
- Escalar Escala el modelo al tamaño dado por el valor s.
- Rotar Rota el modelo en los valores dados para x,y,z.
- Textura Aplica una textura al modelo dado por el valor de t.
- Zoom Aleja o acerca la cámara del modelo con respecto al valor c.

### 4.1.2. Capa del contenedor

Esta capa es la encargada de almacenar los objetos visuales que serán creados en tiempo de ejecución. Una de sus principales características es el arreglo de objetos visuales. Este arreglo permitira mantener el identificador de los objetos visuales desplegados en el cluster de visualización. El servidor de visualización genera una replica a escala de los objetos visuales del cluster de visualización, estos objetos se almacenan en un arreglo. El arreglo del servidor de visualización y del cluster de visualización debe mantener una congruencia en todo momento (ver figura 4.8).

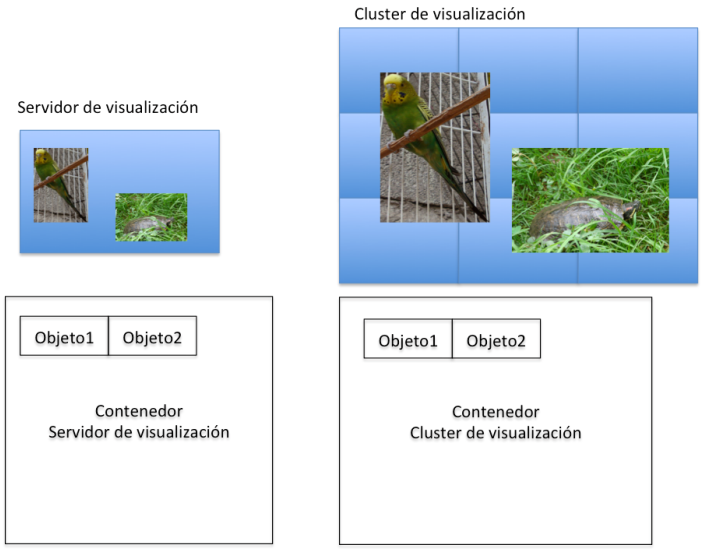


Figura 4.8: Capa del contenedor

Cuando se genera un nuevo objeto visual se almacena en el arreglo con alguna posición inicial. Esta posición mantiene unas coordenadas globales que representan su posición en la pared de video. Cuando el objeto visual cambia su posición se debe actualizar esta posición, el contenedor debe enviar la información sobre el identificador del objeto que realizó la tarea, el reconocedor de eventos debe obtener la información de lo sucedido, y transmitir la información al control. El control se encargará de generar el mensaje con la información necesaria para replicar la tarea en el cluster de visualización.

Las funciones básicas definidas en la API de esta capa se muestran en la figura 4.9. Estas funciones permiten la creación de los objetos visuales después de la inicialización. Esta capa es la encargada de la gestión de los objetos visuales de modo general.

#### Clases de la capa contenedor

Clase Contenedor.- Esta clase es la encargada de almacenar los objetos que son desplegados en pantalla, al menos debe haber un objeto de este tipo en una aplicación. Los objetos

| API de la capa contenedor | | | |
|---|---|---|---|
| Nombre | Parametros | Descripción | Error |
| DVOCrearVisual | File( archivo con alguno de los formatos autorizados) | Esta función permite la creación de un objeto visual dentro de un contenedor desplegado. El despliegue de diversos tipos de formatos para video (3GP, mov, mpg); para imágenes (JPG, PNG, BMP); para modelos 3D archivos con vertices descritos en formatos (.vl con las coordenadas x,y,z de cada punto del modelo). | 121.-El formato de dato inicial no coincide con ningún formato permitido.<br>122.-Error al leer el archivo ó archivo dañado.<br>123.-No se ha iniciado el servidor o los nodos.<br>124.-La conexión entre servidor y nodos es incorrecta. |
| DVOBuscarVisual | Tag (Identificador numérico del objeto visual) | Esta función permite obtener un objeto visual mediante su identificador numérico, regresa el apuntador del objeto. Puede ser utilizado para hacer modificaciones sobre objetos existentes. | 221.- No se encontro ningún objeto con el tag indicado. |
| DVOEliminarVisual | Tag (Identificador numérico del objeto visual) | Esta función permite la eliminación de un objeto visual desplegado dentro de un contenedor, cuando un contenedor es eliminado todos sus objetos visuales son eliminados. | 321.- No se encontro ningun objeto con el tag indicado. |
| DVOPosicionarVisual | Tag (Identificador numérico del objeto visual) | Esta función permite modificar la posición de un objeto visual. | 421.- La posición requerida no se encuentra dentro del contenedor asignado. |

Figura 4.9: API de la capa contenedor

de este tipo permiten el almacenamiento de un objeto visual o de un grupo dentro de un objeto contenedor anidado. Cuando se tiene objetos anidados los objetos observadores deben ser creados en el mismo orden mantenido por los contenedores. De modo que el ultimo elemento de los objetos visuales pueda ser observado por los elementos de la capa de reconocimiento.
Métodos de la clase

- Draw Visualiza el contenido del dato en la dirección (id), en la posición (position) del tamaño (Size), en el visual (tag).
- Add Agrega un nuevo elemento al arreglo que contiene los (tag) de los objetos visuales.
- Delete Elimina el objeto visual con el valor (tag).
- Show Selecciona el valor del elemento visual con el valor (tag).

### 4.1.3. Capa de control

La capa de control recibe la información del usuario y la convierte en mensajes que los nodos pueden reconocer. Se emplea el uso de métodos remotos para cuando los mensajes

involucran eventos de usuario estándar (con mouse y teclado). Esta capa debe estar en el lado del servidor y de los nodos. Del lado del servidor se encarga de buscar la estrategia adecuada para enviar sus paquetes mediante la capa de comunicación. Del lado de los nodos se debe identificar el tipo de mensaje recibido para realizar la acción adecuada. Dependiendo del tipo de dato inicial se elige el algoritmo de distribución de datos.

Esta capa se encarga de conectar la información de las demás capas. Obtiene información del contenedor y del reconocedor de eventos, para generar el mensaje que se envía al cluster de visualización. Esta capa emplea a la capa de comunicación para el envío de mensajes.

Envia el mensaje con la información necesaria para la replica del evento en el cluster de visualización (ver figura 4.10).

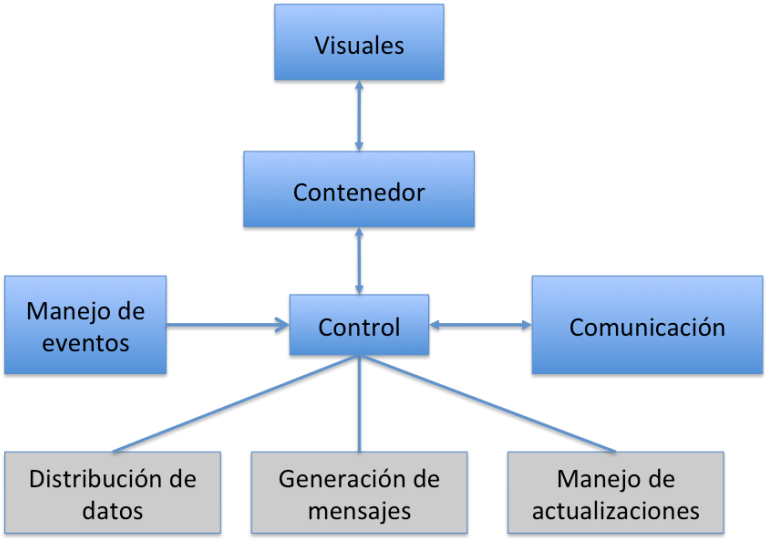


Figura 4.10: Funcionalidad de la capa de control

En esta capa es donde se definen los tiempos de sincronización de despliegue. De modo que recibe los mensajes de todos los nodos y envía el retraso de cada uno con respecto al tiempo de su propio envío. Cada nodo recibe el tiempo que tiene que esperar para desplegar sincronizadamente con los demás. En condiciones óptimas de red, este tiempo debe permanecer constante, de otro modo hay que calcular el retraso cada envío de datos. Cuando el envío contiene la información de un evento de usuario el calculo del retraso para la sincronización es mínimo ya que el paquete esta formado por unos cuantos enteros. En esta capa es donde se emplearán los patrones de diseño strategy y observer. El primero para elegir el algoritmo para la distribución de los datos y el segundo para saber cuando algún cambio ocurra dentro de los objetos visuales.

Las funciones básicas definidas en la API de esta capa para el servidor se muestran en la figura 4.11 y en el nodo en la figura 4.12. Estas funciones permiten la interacción del

lado del servidor con los nodos, esta capa es la encargada del manejo de estrategias de distribución de datos.

| API de la capa de control del lado del servidor | | | |
|---|---|---|---|
| Nombre | Parámetros | Descripción | Error |
| DVORecibeEvento | Origen,posición x,y, tipo de evento | Esta función recibe la información de los eventos recibidos, localiza la posición de origen y re envía la información a los nodos. | 111.-Error al recibir el evento. |
| DVOEnviaDuplicado | | Esta función envía la información a los nodos. | 211.-Error con la conexión. |
| DVOSeleccionaTipo | Archivo de dato inicial | Esta función permite reconocer el tipo de objeto visual que será creado. | 311.-El formato del archivo no es de un tipo reconocido |
| DVOObtenDato | Tipo, Archivo de dato inicial | Toma la información del archivo y la convierte en un dato serializado. | 411.-Error al abrir el archivo. |

Figura 4.11: API de la capa control del lado del servidor

| API de la capa de control en el lado del Nodo | | | |
|---|---|---|---|
| Nombre | Parametros | Descripción | Error |
| DVORecibeDatoEvento | Arreglo [ ] | Esta función recibe un arreglo de cadenas con la información del evento ocurrido en el servidor. | 130.- Error al recibir los datos del evento. |
| DVOReplicaEvento | Arreglo [ ] | Esta función replica el evento dentro del nodo empleando la información recibida. | 230.- Error al recibir los datos del evento. |
| DVOCreaVisualenContenedor | Dato almacenado (apuntador) | Esta función es utilizada para replicar la creación de un objeto visual en el nodo. | 530.- Error en el dato almacenado. |
| DVORecibeDatoEstrategia | TipoEstrategia (integer), dato, arg[ ](cadenas con información sobre posición y tamaño) | Esta función obtiene el dato empleando la estrategia especificada, posteriormente es empleado en la creación del objeto visual con la información obtenida en arg[ ]. | 630.- Error en el tipo de estrategia requerido. |

Figura 4.12: API de la capa control del lado del nodo

La figura 4.13 muestra las estrategias de distribución de datos permitidas en esta capa. Estas estrategias varian la forma en la que el dato es recibido en el nodo y enviado desde el servidor.

| Estrategias de distribución | | |
|---|---|---|
| Estrategia | Valor | Descripción |
| Replicación | 1 | Esta estrategia envia la información completa a cada nodo, donde posteriormente es desplegada. |
| Fragmentación | 2 | Esta estrategia envia porciones de información a cada nodo, donde posteriormente es desplegada. |
| Replicación-fragmentación | 3 | Esta estrategia replica porciones de información en los nodos, donde es desplegada la información. |

Figura 4.13: Estrategias de distribución de datos

**Clases de la capa de control**

Clase Strategy.- Esta clase es la encargada de recibir el dato inicial y de visualizarlo dependiendo del tipo indicado. Esta clase es general para cualquier tipo de objeto, cuando se reconoce el tipo específico de objeto se llaman a las clases específicas ImageStrategy, VideoStrategy y ModelStrategy.
Métodos de la clase

- Open Este método es el encargado de obtener el dato desde el archivo para posteriormente reconocerlo y dependiendo de su tipo iniciar la distribución a los nodos de despliegue.
- Distribution Es el método encargado de seleccionar el método de distribución correcta para el tipo de dato dado.

Clase ImageStrategy.- Esta clase es la especificación de la clase Strategy y es la encargada de recibir imágenes y visualizarlas.
Métodos de la clase

- Open Este método es el encargado de obtener las imágenes desde el archivo para posteriormente iniciar la distribución a los nodos de despliegue.
- Distribution Es el método que genera la distribución para las imágenes.

Clase VideoStrategy.- Esta clase es la especificación de la clase Strategy y es la encargada de recibir los videos y visualizarlos.
Métodos de la clase

- Open Este método es el encargado de obtener los vídeos desde los archivo para posteriormente iniciar la distribución a los nodos de despliegue.
- Distribution Es el método encargado de generar la distribución para el vídeo.

Clase ModelStrategy.- Esta clase es la encargada de recibir el archivo con las coordenadas iniciales del objeto 3D y de graficarlas en pantalla.
Métodos de la clase

- Open Este método es el encargado de obtener el dato desde el archivo para posteriormente iniciar la distribución a los nodos de despliegue.
- Distribution Es el método encargado de la distribución correcta del modelo.

### 4.1.4. Capa de comunicación

La capa de comunicación se relaciona con el manejo de red del sistema operativo. Se encarga de la conexión de la máquina cliente con el servidor, el envío de mensajes y manejo de métodos remotos; los cuales permiten el manejo de los objetos visuales de modo remoto. Con este principio es posible permitir que los nodos reciban la información desde el servidor. Este objeto esta fundamentado en el concepto de objeto distribuido y su modelo es similar, con el cambio de enfoque hacia un objeto dedicado al despliegue.

*Objetos Distribuidos Visuales (DVO). El objeto distribuido visual se define como un objeto visual que permite interacción remota con otros objetos visuales.* (ver figura 4.14)

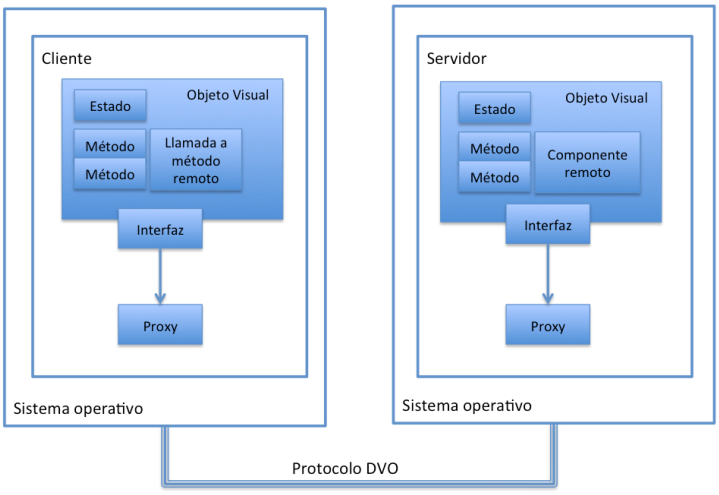


Figura 4.14: Modelo del objeto distribuido visual

Las invocaciones remotas que se ejecutan entre los objetos visuales están definidas en el protocolo DVO. El protocolo DVO 4.15 define los métodos remotos almacenados en los objetos visuales del cluster de visualización. Una de las opciones viables que permiten

la implementación de este protocolo es mediante el uso de un proxy. Algunos lenguajes implementan esta funcionalidad, incluido Objetive-C.

| Protocolo DVO | | |
|---|---|---|
| Método | Parámetro | Descripción |
| DVORecibeDatoEvento | Arreglo[ ] | Este método toma la información del arreglo [ ] para replicar el evento con los objetos del nodo. |
| DVORecibeDatoEstrategia | TipoEstrategia (integer), dato, arg[ ](cadenas con información sobre posición y tamaño). | Este método recibe el dato inicial y su tipo para crear la replica del objeto visual en el nodo. |

Figura 4.15: Protocolo DVO

Dentro de las capas OSI buscamos una opción en la capa de transporte que permita el control de los mensajes y además este orientado a mensajes es por eso que se eligió el protocolo SCTP (ver figura 4.16).

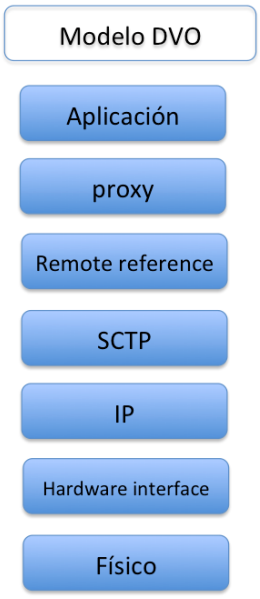


Figura 4.16: Capas OSI (DVO)

En esta capa se emplea la notificación para el envío de mensajes a todos los nodos y la invocación remota cuando el mensaje es enviado a un sólo nodo. El uso de los objetos distribuidos emplea el estado replicado cuando se utiliza el modo con múltiples ventanas y el fragmentado en el caso del modo de pantalla completa. En esta capa el patrón de diseño

que se considera es *state* , como se comenta en el capítulo de marco teórico, el patrón state permite a un objeto alterar su comportamiento cuando ocurran cambios internos de estado. Por lo que el uso de este patrón nos permite controlar los diferentes estados de la conexión entre el servidor y el cluster de visualización.

Las funciones básicas definidas en la API de esta capa se muestran en la figura 4.17. Estas funciones permiten la creación de los objetos empleados en la comunicación. Estas funciones permiten el envío de mensajes entre el servidor y los nodos.

| API de la capa de comunicación | | | |
|---|---|---|---|
| Nombre | Parametros | Descripción | Error |
| DVOEnviaRMI | Dato<br>Tipo<br>IDConexionRMI | Esta función envia los parametros Dato y Tipo usando la conexion IDConexionRMI previamente creada. | 141.- Error de conexión<br>142.- Error con los formatos de los parámetros Dato y Tipo. |
| DVORecibeRMI | Dato<br>Tipo | Esta función es la encargada de recibir la información del lado de los nodos. Estas funciones se definen en el protocolo DVO, en ambos nodos y servidor. | 241.- Error de conexión<br>242.- Error con los formatos de los parámetros Dato y Tipo. |
| DVOCrearConexionRMI | IpMaquina<br>PuertoMaquina | Esta función se encarga de realizar la conexión entre los nodos y el servidor mediante el protocoloDVO definido. Regresa el identificador de la nueva conexión. | 341.- No se pudo establecer la conexión.<br>342.- Puertos ocupados. |
| DVOCrearConexionNotificacion | IpMaquina<br>PuertoMaquina<br>Método | Esta función habilita los observadores en las IP's y puertos requeridos para el método asignado. | 441.- Las teclas requeridas han sido asignadas previamente. |
| DVOLiberaConexionRMI | ConexionRMI | Esta función libera la memoria de las variables utilizadas en la conexión. | 541.- Las variables están siendo utilizadas. |
| DVOLiberaConexionNotificación | ConexionNotificacio n | Esta función libera la memoria de las variables utilizadas en la conexión. | 641.- Las variables están siendo utilizadas. |
| DVONotifica | Dato<br>Tipo | Esta función envia el Dato y tipo a los observadores. | 741.- Error de conexión<br>742.- Error con los formatos de los parámetros Dato y Tipo. |

Figura 4.17: API de la capa de comunicación

### Clases de la capa de comunicación

Clase Comunicación.- Esta clase es la encargada de habilitar la comunicación entre los nodos y el servidor, independientemente del modelo de comunicación empleado posteriormente. En esta clase se obtiene la información de configuración inicial necesaria para la comunicación, es decir puertos e IPs de cada máquina. Una vez iniciada la conexión con algunos de los tipos de comunicación este objeto no podrá modificar ningún elemento de sus arreglos de IPs y puertos.
Métodos de la clase

- GetIPOn(int tag) Obtiene la IP almacenada en la posición (tag) en el arreglo de (IP) que es creado desde la configuración inicial y obtenido del archivo (ConfigIn).
- PutIPOn(int tag) Coloca el valor de la IP en la posición (tag) en el arreglo de (IP).
- GetPuertoOn(int tag) Obtiene el valor del puerto en la posición (tag) en el arreglo de (Puertos) que es creado desde la configuración inicial y obtenido del archivo (ConfigIn).
- PutPuertoOn(int tag) Coloca el valor del puerto en la posición (tag) en el arreglo de (Puertos).

Clase MPI .- En esta clase se realizan las inicializaciones necesarias para realizar el paso de mensajes, ya que este tipo de comunicación no es orientado a la conexión, cada envío implica una nueva inicialización.
Métodos de la clase

- Send(arg1,arg2) Dada la inicialización del objeto comunicación los argumentos (arg1y arg2) son enviados a las (IPs) y (Puertos) del arreglo dado por la configuración inicia.
- Compact(data) Realiza la conversión de los argumentos dependiendo del tipo (type) de dato que será enviada como argumento por el método send.

Clase Notificación.- En el caso de esta clase es necesario agregar una lista de los elementos que serán notificados. Esta inicialización se hace en una sola ocasión.
Métodos de la clase

- IniciaNotificados La información de los elementos que serán notificados son obtenidos de los arreglos IP y Puertos.
- Send(arg1,arg2) Dada la inicialización del objeto comunicación los argumentos (arg1y arg2) son enviados a las (IPs) y (Puertos) del arreglo dado por la configuración inicia.
- Compact(data) Realiza la conversión de los argumentos dependiendo del tipo (type) de dato que será enviada como argumento por el método send.

Clase UDP.- Este tipo de envío no es orientado a conexión por lo que dentro del método (send) es donde se crea el socket tipo UDP y se envía la información. Para esto se debe definir el algoritmo de envío de bytes mediante el método (StreamByte).
Métodos de la clase

- StreamByte Se encarga de definir la cantidad de bytes enviados en cada paquete a ser enviado.
- Send(arg1,arg2) Dada la inicialización del objeto comunicación los argumentos (arg1y arg2) son enviados a las (IPs) y (Puertos) del arreglo dado por la configuración inicia.

- Compact(data) Realiza la conversión de los argumentos dependiendo del tipo (type) de dato que será enviada como argumento por el método send.

### 4.1.5. Capa de reconocimiento de eventos

Para fines de simplicidad en esta tesis las acciones y los eventos se consideran iguales. Las tareas que definiremos a continuación son iniciadas por eventos de usuario y pueden involucrar uno o más eventos. Como lo indica el libro [Lauesen, 2005] en el capítulo 15, en HCI algunas tareas inician con un evento, y todos los eventos deben ser manejados por tareas, ya sean manual o automáticamente.

En este modelo la capa de manejo de eventos de usuario permite la interacción del usuario con las aplicaciones finales. Existen diversos modos para reconocer el evento, uno de ellos es mediante el uso de tablas de reconocimiento. La ventaja de usar tablas para reconocer los eventos es que agregando elementos a la tabla podemos identificar nuevos eventos (ver figura 4.18). Dentro del manejo básico proporcionado en este modelo para el contenedor se proponen las siguientes tareas: (1) la movilidad de la ventana, (2) cambio de tamaño, (3) selección y (4) eliminación de objetos visuales.

| Dictionario general (Del contenedor) | Acción a realizarse |
|---|---|
| Arrastre de ratón pulsando botón derecho | Cambia la posición del objeto visual |
| Arrastre de ratón pulsando botón izquierdo | Cambia el tamaño del objeto visual |
| Ctrl+delete | Borra el objeto visual seleccionado |
| Doble click | Selecciona un objeto visual |

Figura 4.18: Manejo de eventos del (DVO)

La información generada sobre el evento ocurrido es tomada de la cola de eventos del sistema operativo. El manejo de eventos (Ver Figura 4.19), se comunica directamente con el servidor de ventanas del sistema operativo. Se encarga de recuperar la información del tipo de evento, dependiendo de su procedencia y su localización. Después de reconocer el evento, se envía un mensaje con la información a los nodos.

La arquitectura de esta funcionalidad se puede ver de manera horizontal iniciando cuando el evento ocurre. La información se obtiene desde el sistema operativo y pasa a la cola de eventos. De la cola de eventos llega a la aplicación y a la ventana. Después es enviada al contenedor y de ahí al objeto visual. Una vez que se generó el mensaje con el evento ocurrido, debe ser enviado a los servidores para ser ejecutado en el objeto visual.

#### Clases de la capa de reconocimiento de eventos

Clase ObserverSubject.- Esta clase se encarga de obtener la información generada cuando un evento de usuario ocurre.
Métodos de la clase

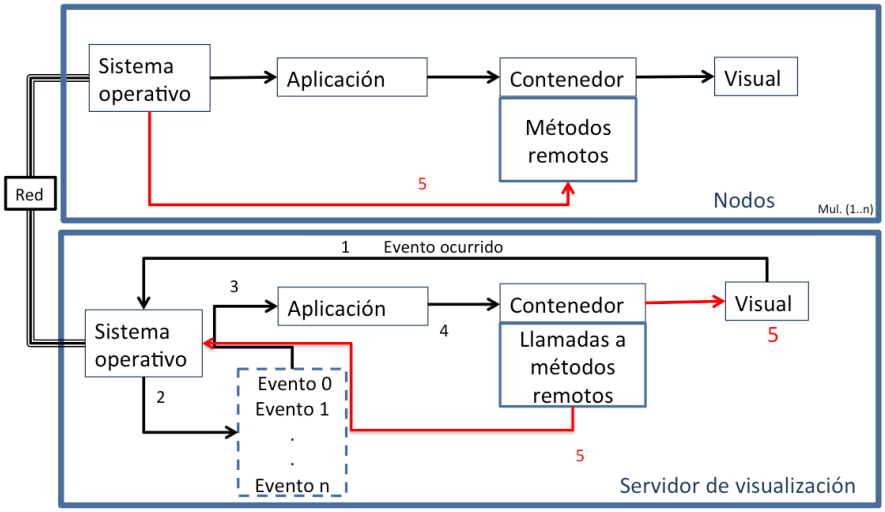


Figura 4.19: Manejo de eventos del (DVO)

- GetSubject Obtiene el identificador del objetivo con el valor (tag) en el arreglo de objetivos (arraysubject).
- PutSubject Agrega un nuevo objetivo al arreglo de, esto ocurre cada vez que un nuevo objeto visual es creado.
- GetState Obtiene el estado del objetivo con el valor (tag).
- PutState Forza el cambio de estado del objetivo con el valor (tag).

## 4.2. Formalización del modelo

La formalización de los objetos distribuidos visuales (DVO) nos ayudan a validar algunas de las propiedades del modelo DVO. Iniciamos con las definiciones conocidas del paradigma orientado a objetos y posteriormente obtendremos la descripción formal de nuestros objetos.

### 4.2.1. Definiciones del paradigma orientado a objetos

A continuación exponemos las propiedades del paradigma orientado a objetos. Estas definiciones forman parte de la base que nos ayudará a validar las definiciones posteriores.

DEFINICIÓN **1 (Abstracción)** *Sea $C$ el conjunto de las características fundamentales de algo, tal que:*

$\nexists$ *antecedentes de* $C \wedge \nexists$ *detalles de* $C$.

DEFINICIÓN **2 (Encapsulación)** *Sea* $o \in O$ *que implementa a una abstracción* $C$ *que constituye su estructura y su comportamiento tales que:*

- *Oculta la información.- o revela lo menos posible su estructura interna.*
- *Parte pública.- o permite la manipulación de su estructura interna únicamente mediante su parte pública.*

DEFINICIÓN **3 (Modularidad)** *Sea* $S$ *un sistema descompuesto en un conjunto de elementos* $o$*, tal que:*

- *Cohesivos.- S es una agrupación de conjuntos tipo C de abstracciones que guardan una relación lógica.*
- *Débilmente acoplados.- Los elementos de S utilizan las mínimas dependencias entre ellos.*

DEFINICIÓN **4 (Jerarquía)** *Sea* $j$ *una jerarquía, tal que mantiene una clasificación de las abstracciones* $C$*.*

- *Herencia.- Define una relación donde la clasificación de las abstracciones C mantiene una relación de generalización y especialización, tal que:*
  - *Herencia múltiple.- Define una relación entre las abstracciones, en las que una abstracción comparte la estructura de comportamiento definida en más de una abstracción.*
  - *Herencia simple.- Define una relación entre las abstracciones, en las que una abstracción comparte la estructura de comportamiento definida en una sola abstracción.*
- *Composición.- Define una relación donde la clasificación de las abstracciones C mantienen una relación de agregación.*

DEFINICIÓN **5 (Polimorfismo)** *Sea* $c_1, c_2 \in C$ *una abstracción definida por un estado y un comportamiento, de modo que pueden existir* $c_1$ *y* $c_2$ *con el mismo nombre y con comportamiento diferente.*

### 4.2.2. Concepto de objeto

En esta sub sección se presentan los conceptos que definen a los objetos. Sus componentes principales y la caracterización de las propiedades anteriores empleando el concepto de clases.

DEFINICIÓN **6 (Objeto)** *Dado un conjunto de atributos $\alpha$ , sobre el que se ha definido un conjunto de métodos $\mu$, que domina implícitamente a $\alpha$, unido a un identificador único $i$, se dice que $o$ es un objeto definido como una tupla $(\mu, \alpha, i)$.*

DEFINICIÓN **7 ( Atributos )** *Sea $\alpha$ un conjunto dado por :*

$$\alpha = \{a_1, a_2, ..., a_n\}$$

*de modo que, por ejemplo $a_1$ puede ser alguno de los siguientes tipos:*

$$a_1 = \begin{cases} integer \\ char \\ double \\ object \\ \dots \end{cases} \tag{4.1}$$

DEFINICIÓN **8 (Métodos)** *Sea $\mu$ un conjunto, elemento de la tupla $(\mu, \alpha, i)$ que definen un objeto $o$ tal que:*

$\forall \mu_j \in \mu, \alpha_r \in \alpha : \exists \mu_j (o) = \alpha_r$

DEFINICIÓN **9 (Clase)** *Dado $c \in C$, donde $c$ es la definición abstracta de un grupo de objetos $o$, tal que $c$ se diferencie de otras clases por su tupla $(\mu, \alpha)$ y cumpla con las propiedades del paradigma orientado a objetos.*

- *Jerarquia (Herencia simple).- Sean $c_h, c_m \in C$ donde $c_h, c_m$ están definidos por las tuplas $(\mu_h, \alpha_h)$,$(\mu_m, \alpha_m)$ respectivamente, decimos que $c_h$ hereda de $c_m$ si se cumple que:*

  $(\exists m \in \mu_h : m \in \mu_m) \vee (\exists a \in \alpha_h : a \in \alpha_m)$

- *Jerarquia (Herencia multiple).- Sean $c_h, c_k(k : 1..n) \in C$ donde $c_h, c_r$ están definidos por las tuplas $(\mu_h, \alpha_h)$,$(\mu_k, \alpha_k)$ respectivamente, decimos que $c_h$ hereda de $c_k(k : 1..n)$ si se cumple que:*

  $(\exists m \in \mu_h : m \in \bigcup_{k=1}^{k=n} \mu_k) \vee (\exists a \in \alpha_h : a \in \bigcup_{k=1}^{k=n} \alpha_k)$

- *Composición.- Sean $c_1, c_2 \in C$ donde $c_1, c_2$ están definidos por las tuplas $(\mu_1, \alpha_1)$,$(\mu_2, \alpha_2)$ respectivamente, se dice que $c_1$ esta compuesto por $c_2$ siempre que:*

  $c_2 \in \alpha_1$

- *Clase abstracta.- Sea $c \in C$ definida como la tupla $(\mu, \alpha)$, tal que:*

  $\nexists$ *o definido por c, a menos que* $\exists c_1$ *que herede de c, donde o implementa a* $c_1$.

- *Polimorfismo.- Sean $c_1, c_2 \in C$ definidos como las tuplas $(\mu_1, \alpha_1)$,$(\mu_2, \alpha_2)$ respectivamente, tal que:*

  *El nombre de* $c_1$ *y* $c_2$ *es el mismo y* $\mu_1 \neq \mu_2$.

  DEFINICIÓN **10 (Interfaz)** *Sean $o_1$ elemento del conjunto de objetos O, donde $o_1$ está definido por la tupla $(\mu_1, \alpha_1, i_1)$ definimos el subconjunto de métodos de la interfaz $\mu_{i1}$, como $\mu_{i1} \subseteq \mu_1$ tal que:*

  $\forall a_1 \in \alpha_1 : \exists!$ *acceso a* $a_1$ *mediante elementos del subconjunto* $\mu_{i1}$
  *De modo que:*

  $\forall o \in O : \exists!$ *acceso a* $\alpha_1$ *de* $o_1$ *mediante elementos del subconjunto* $\mu_{i1}$.

### 4.2.3. Premisas de objetos específicos

Premisas necesarias en las definiciones de los objetos remoto, distribuido y visual.

DEFINICIÓN **11 (Aplicación)** *Sea d una aplicación definida por la tupla $(\beta, \gamma, \kappa)$, donde $\beta$ es el conjunto de los objetos implementados en $d, \gamma$ es el conjunto de direcciones de memoria asignados a los objetos y $\kappa$ el conjunto de mensajes enviados entre los objetos, tal que:*

*Si* $o_1, o_2 \in \beta \rightarrow o_1, o_2 \in \gamma$

*Si* $o_1, o_2 \in \gamma \rightarrow \exists g \in \kappa : g(o_1 \rightarrow_( mensaje) o_2)$

DEFINICIÓN **12 (Mensaje Remoto)** *Dados los objetos $o_1, o_2$ definidos por las tuplas $(\mu_1, \alpha_1, i_1)$, $(\mu_2, \alpha_2, i_2)$, con una interfaz $\mu_{i1} \subseteq \mu_1$, ubicados en las aplicaciones $d_1, d_2$ definidas por las tuplas $(\beta_1, \gamma_1, \kappa_1)$,$(\beta_2, \gamma_2, \kappa_2)$ respectivamente, se dice que $d_1$ envia un mensaje remoto a $d_2$ siempre que:*

$\exists o_1 \in \beta_1 \wedge \exists o_2 \in \beta_2 : (\exists m \in \mu_{i1}$*: m es usado en* $\mu_2)$

DEFINICIÓN **13 (Tamaño de un Mensaje Remoto de datos iniciales)** *Dado un m mensaje remoto de una aplicación $d_1$ a $d_2$, entre dos objetos $o_1, o_2$ definidos por las tuplas $(\mu_1, \alpha_1, i_1)$, $(\mu_2, \alpha_2, i_2)$. Definimos una función:*
$T(m) = a_1$, *donde* $a_1 \in \alpha_1$.

DEFINICIÓN **14 (Protocolo)** *Dados los objetos $o_1, o_2$ definidos por las tuplas $(\mu_1, \alpha_1, i_1)$, $(\mu_2, \alpha_2, i_2)$, con una interfaz $\mu_{i1} \subseteq \mu_1$, ubicados en las aplicaciones $d_1, d_2$ definidas por las tuplas $(\beta_1, \gamma_1, \kappa_1)$,$(\beta_2, \gamma_2, \kappa_2)$, definimos un protocolo $p_{o1o2} \subset \mu_{i1}$, donde $o_1 \in d_1 \wedge o_2 \in d_2$ tal que:*
$\forall m \in p_{o1o2}$*:* $m \in \mu_2$

DEFINICIÓN **15 (Pantalla)** *Sea p un despliegue definido por una matriz de puntos $MP$ de tamaño $(p_{r_x}, p_{r_y})$, controlado por una aplicación $d_1$ definida por la tupla $(\beta_1, \gamma_1, \kappa_1)$, donde:*
$\forall$ *punto* $\in MP_{p_{r_x}, p_{r_y}}$*:* $\exists$ $o \in \beta_1$ *que le proporciona la información.*

DEFINICIÓN **16 (Cuadro)** *Sea c una área rectangular definida por una matriz de puntos $MP$ de tamaño $(c_x, c_y)$, con una posición inicial definida por $(op_x, op_y)$ sobre una pantalla p controlada por una aplicación ddefinida por la tupla $(\beta, \gamma, \kappa)$, in donde:*
$\forall$ *pixel* $\in MP_{c_x, c_y}$*:* $\exists$ $o \in \beta$ *que le proporciona la información.*

DEFINICIÓN **17 (Pared de vídeo)** *Sea pv un conjunto de despliegues p puestos en nx,ny posición, donde cada p esta definido por una matriz de puntos $MP$ del mismo tamaño $(p_{r_x}, p_{r_y})$, donde el tamaño de pv está dado por:*
$\bigcup_{k=1}^{k=n} p$ *donde:*
$pv_(r_x) = \sum_{k=1}^{k=nx} p_{r_x} \wedge pv_(r_y) = \sum_{k=1}^{k=ny} p_{r_y}$.

DEFINICIÓN **18 (Objeto Distribuido)** *Definimos a $DO \subseteq O$ donde $\forall o \in DO$ se cumple lo siguiente:*

*Dados dos objetos $o_1, o_2 \in O$ definidos por las tuplas $(\mu_1, \alpha_1, i_1)$,$(\mu_2, \alpha_2, i_2)$, ubicados en las aplicaciones $d_1, d_k(k : 1..n)$, con un protocolo definido por $p_{o1o2}$, decimos que $o_1 \in DO$ siempre que:*
$\forall m \in p_{o1o2}$*:* $m \in \mu_2 \wedge \exists a \in \alpha_1$*:* $a \in d_k{}_{k=1}^{k=n}$

DEFINICIÓN **19 (Objeto Visual)** *Definimos a $VO \subseteq O$ donde $\forall o \in VO$ se cumple lo siguiente:*

*Dado un $o \in VO$ definido por la tupla $(\mu, \alpha, i)$ en una aplicación d definida por la tupla $(\beta, \gamma, \kappa)$ que controla un despliegue tipo cuadro c de tamaño $(c_x, c_y)$. El objeto o cumple con lo siguiente:*

- $(\exists$ $m \in \mu$*:* *m realiza el despliege en c* $) \wedge (\exists$ $a \in \alpha : \forall$ *punto*$_{c_x, c_y}$ *hay dato).*
- $\exists$ $m \in \mu$*:* *m reconoce eventos ocurridos en el área de c.*
- *Se cumple con el anidamiento jerárquico parcial( ver figura 4.20 ) como:*
  *Sea $DES$ un conjunto dado por :*

$$DES = \{d_1, d_2, ..., d_n\}$$

*de modo que $d_1$ es cualquier elemento visual que puede ser desplegado, como lo son:*

$$d_1 = \begin{cases} video \\ imagen \\ modelo3D \\ \dots \end{cases} \tag{4.2}$$

*Con los conjuntos $V, C \subset VO$ [1], sean $o_c \in C$ y $o_v \in V$ definidos por las tuplas $(\mu_c, \alpha_c, i_c)$ y $(\mu_v, \alpha_v, i_v)$ donde:*

*($\forall$ $d \in DES$ : $\exists$ $o_v \in V$: $\mu_v$ realiza el despliegue en $d$ $\wedge \alpha_v$ contiene a $d$ ) $\wedge (\exists o_c \in C : o_v \in \alpha_c)$*

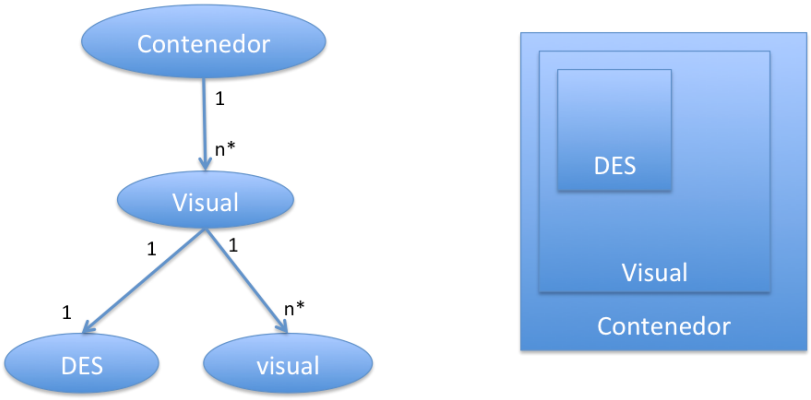


Figura 4.20: Anidamiento jerárquico parcial de objetos visuales

Dadas las definiciones anteriores es posible obtener la definición formal del objeto distribuido visual, el cual es la base del modelo DVO.

DEFINICIÓN **20 (Objeto Distribuido Visual)** *Sean $o_{DO} \in DO, o_{VO} \in VO$ definidos por las tuplas $(\mu_{DO}, \alpha_{DO}, i_{DO})$, $(\mu_{VO}, \alpha_{VO}, i_{VO})$, donde $o_{DO}$ implica la existencia de un protocolo $p_(o_1, o_2)$, definimos $DVO \subset O$ tal que:*

*( $\exists$ $o_{VO} \in \alpha_{DO}$) $\rightarrow o_{DO} \in DVO$. (Definición formal del objeto distribuido visual)*

[1] En la implementación el subconjunto V esta formado por los objetos visuales, C por los contenedores y VO se describe como la clase abstracta de la que heredan.

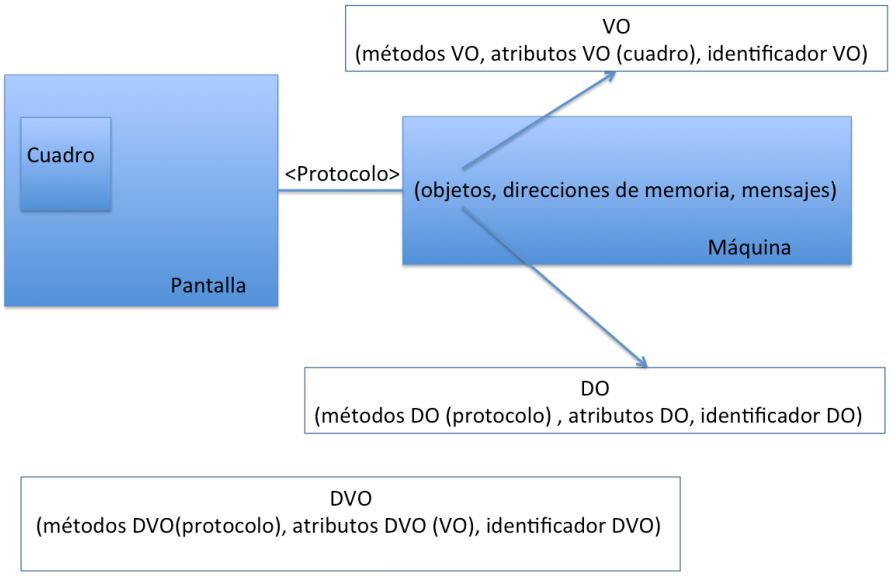


Figura 4.21: Composición de objetos VO y DO para formar el objeto DVO

*La figura 4.21 muestra el comportamiento y posición de los objetos descritos con anterioridad. Esta definición se muestra en la relación uno a uno de una máquina una pantalla, pero puede ser modificada de uno a muchos. De donde se derivan las siguientes propiedades:*

*Dados dos objetos* $o_1, o_2 \in DVO$ *definidos por la tuplas* $(\mu_1, \alpha_1, i_1)$,$(\mu_2, \alpha_2, i_2)$, *sobre las aplicaciones* $d_1, d_2 \in \delta$ *que controlan las pantallas* $p_1, p_2 \in \Phi$. *Tenemos que:*

$\forall o_1, o_2 \in DVO : \exists m_1 \in \mu_1 \wedge m_2 \in \mu_2 :$ *$m_1$ despliega en $p_1 \wedge m_2$ despliega en $p_2$ al mismo tiempo. (Sincronización)*

$\forall o_1, o_2 \in DVO : \exists$ *un protocolo definido* $p_{o1o2}$.

$\forall o_1, o_2 \in DVO : \exists\ a \in \alpha_1$: $a \in \alpha_2 \vee \frac{a}{n} \in \alpha_2$. *Donde $n$ es el número de aplicaciones en las que se encuentra distribuido (Distribución de datos).*

$\forall o_1, o_2 \in DVO : \exists m \in \mu_1 :$ *$m$ reconoce un evento $\wedge o_1$ envia mensaje remoto a $o_2$. (Manejo de eventos)*

El cumplimiento de todas las propiedades de los objetos distribuidos visuales permite realizar el manejo de una pared de vídeo. Manteniendo la independencia de la aplicación final. El uso de la red moderado. La distribución de la carga de trabajo (el despliegue). Así como la construcción de visualizaciones complejas anidadas (que en un futuro pueden

trabajarse como ventanas). De ahí que esta formalización pueda ser empleada para la construcción de un manejador de ventanas distribuido.

TEOREMA **1 (De la disminución del tamaño del mensaje usando DVO)** *Dados dos objetos* $o_{dvo} \in DVO$ *y* $o \in O$ *donde* $\forall o \notin DO$ *entonces:*

$T(m_{dvo}) \leq T(m_o)$.

*El tamaño de un mensaje remoto de dato inicial enviado entre dos aplicaciones* $d_1$ *y* $d_2$ *es menor o igual al tamaño del mensaje remoto de dato inicial usando un* $o \in O$ *sin características de distribución.*

DEMOSTRACIÓN **1 (Teorema 1)** *Dados dos objetos* $o_1, o_2 \in DVO$ *definidos por las tuplas* $(\mu_{DO_1}, \alpha_{DO_1}, i_{DO_1})$,$(\mu_{DO_2}, \alpha_{DO_2}, i_{DO_2})$, *con un protocolo definido* $p_(o_1, o_2)$.

*De la definición de DVO tenemos que* $\forall o_1, o_2 \in DVO : \exists\ a \in \alpha_1$*:* $a \in \alpha_2 \vee \frac{a}{n} \in \alpha_2$. *De modo que cada fracción de* $a_1 \in \alpha_1$ *puede ser enviada como dato inicial. Si empleamos la función de tamaño de mensaje remoto de dato inicial tenemos que para un* $o_{dvo} \in DVO$. *Se cumple lo siguiente*

$$T(m_{dvo}) = \frac{a_1}{n} \vee T(m_{dvo}) = a_1 \tag{4.3}$$

*Empleando la misma definición con un* $o \in O$ *sin características de distribución tenemos que:*

$$T(m_o) = a_1 \tag{4.4}$$

*Por lo que de la ecuación 5.3 y 5.4 tenemos que:*

$T(m_{dvo}) \leq T(m_o)$.

A continuación se muestra un análisis cuantitativo de una de las propiedades del subconjunto DVO (Distribución de datos), empleado en el control de una pared de video $pv \in \Phi$ controlada mediante un grupo de aplicaciones $d \in \delta$ definidas anteriormente, desplegando un dato en particular (imágenes). Para dicho análisis requerimos la siguiente simbología:

- (sTD) tamaño de la matriz *pv*
- (sI) tamaño original de la imagen.
- (nM) número de mensajes requeridos entre las aplicaciones d sin acuse de recibo (ACK).

- (sM) tamaño de los mensajes enviados.
- (nS) número de aplicaciones $d$ que controlan $pv$.
- (nI) número de imágenes enviadas.
- ($nTu_i$) número de pantallas $p \in pv$ usadas por una imagen i.
- ($nV_i$) número de vecinos que rodean las pantallas $p$ empleadas en el despliegue.

Fragmentación de la imagen.- Esta estrategia se basa en la fragmentación de la imagen. Cada fragmento puede tener diferente tamaño. El tamaño de los fragmentos es el mismo en el caso de usar la resolución de la $pv$ completa.

En el caso de los datos iniciales las ecuaciones que definen el tamaño del mensaje son las siguientes:

$$nM = \sum_{i=1..nI} (nTu_i) \tag{4.5}$$

$$sM_i = (sI_i/nTu_i) \tag{4.6}$$

Cuando las imágenes se mantienen en ventanas flotantes las ecuaciones se convierten en las siguientes:

$$nM = \sum_{i=1..nI} (nTu_i) \tag{4.7}$$

$$sM_i = (sI_i/nTu_i) \tag{4.8}$$

Replicación de la imagen.- Cuando la información se encuentra desplegada dentro de diferentes imágenes la replicación de información puede mejorar el desempeño, ya que es posible evitar enviar el paquete nuevamente. Simplemente se puede enviar el identificador del despliegue y la su nueva posición.

En el caso de los datos iniciales las ecuaciones que definen el tamaño del mensaje son las siguientes:

$$nM = nS * nI \tag{4.9}$$

$$sM_i = (sI_i) \tag{4.10}$$

Cuando las imágenes se mantienen en ventanas flotantes las ecuaciones se convierten en las siguientes:

$$nM = nS * nI \tag{4.11}$$

$$sM_i = constant \tag{4.12}$$

Capítulo 5

# Caso de estudio: Visor de imágenes

En este capítulo tomamos los conceptos explicados en el modelo DVO para aplicarlos en un prototipo específico. Este prototipo tiene como finalidad el despliegue de imágenes de alta resolución. Además se incluyen las especificaciones del sistema y su funcionalidad básica.

Como se comenta en el capítulo del estado del arte, el uso de las paredes de vídeo es básico en diversos laboratorios. Esto ha generado numerosas investigaciones, de ahí surgen diversas bibliotecas, frameworks y middlewares que permiten trabajar con paredes de vídeo. Cada una de ellas proporciona sus beneficios pero también genera sus propios requerimientos. Por ejemplo, en el grupo de investigación de SAGE se logro permitir el despliegue de distintos tipos de aplicaciones sobre la pared de vídeo, mediante el uso de grandes cantidades de red (10 gb). Es por eso que en este prototipo proporcionamos el desarrollo de un visor de imágenes (basado en el modelo DVO) que trabaja sobre una pared de vídeo (CinvesWall) y funciona con elementos mínimos en hardware (a bajo costo comparado con paredes de este tamaño). Pero que a su vez permite escalabilidad en caso necesario.

La abstracción jerárquica del Cinveswall (ver figura 5.1), permite delimitar los cuatro bloques principales; vistos en el modelo DVO. Las metas de alto nivel, las prioridades y restricciones, funcionalidades generalizadas y la función física; las metas de alto nivel que se verán reflejadas en las aplicaciones del usuario son básicamente el despliegue de imágenes, vídeos y modelos 3D, en una resolución total de 2600x7680 de una matriz de pantallas 3x4 con 1920x1200 pixeles de resolución cada una.

El prototipo usa el modelo cliente-servidor, el cual funciona con un cluster de visualización, un servidor de visualización y las pantallas de despliegue (ver figura 5.2). Sobre el servidor de visualización se encuentra la interfaz de usuario, de esta manera se permite un manejo más sencillo para usuarios familiarizados con escritorios estándar.

Nuestra aplicación usa el menor número de entradas posible, a fin de facilitar el trabajo para usuarios poco expertos en los aspectos de configuración de estos ambientes. Además facilitamos una serie de herramientas que permitan a un usuario experto incrementar controles dentro de la interfaz. Es por eso que se pensó en un framework DVO, que puede

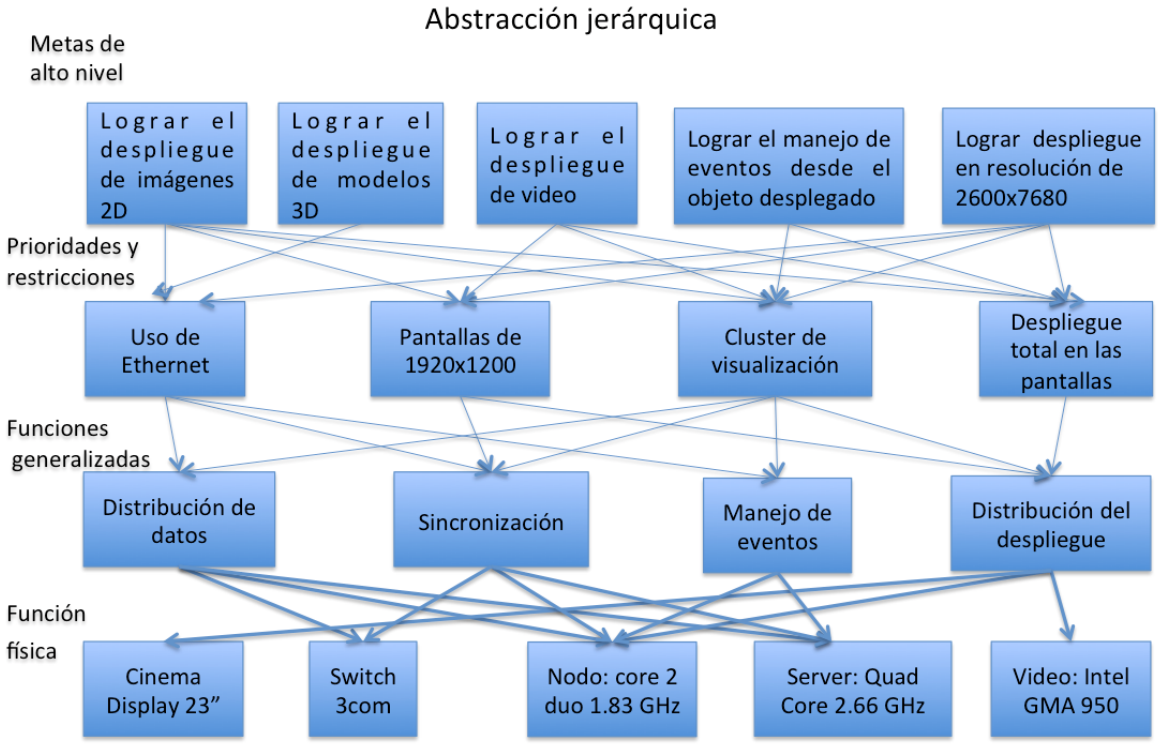


Figura 5.1: Abstracción jerárquica del problema(Cinveswall)

ser usado para crear y modificar métodos de los objetos desplegados.

La interfaz de usuario muestra una ventana con dos botones, el primer botón es para abrir una nueva imagen mediante un cuadro de diálogo. El segundo botón permite entrar en los detalles de configuración; los cuales están definidos por default para facilitar su uso. Cuando la imagen esta desplegada en la pantalla el usuario puede moverla mediante el arrastre con el botón derecho del mouse.

Los aspectos de configuración modificables en tiempo de ejecución son los relacionados con el aspecto de las imágenes que se abrirán. Los aspectos de redes y manejo de eventos solo pueden modificarse antes de iniciar el sistema.

## 5.1. Caracterización del modelo DVO en un Visor de imágenes

El funcionamiento del sistema completo es el siguiente: el servidor de visualización muestra una ventana con dos botones (abrir y configuración), al mismo tiempo las pantallas del cluster de visualización se muestra el fondo del mismo color que en el servidor. A partir de la inicialización el servidor y las pantallas mostrarán la información al mismo tiempo, a una escala diferente(ver figura 5.3).

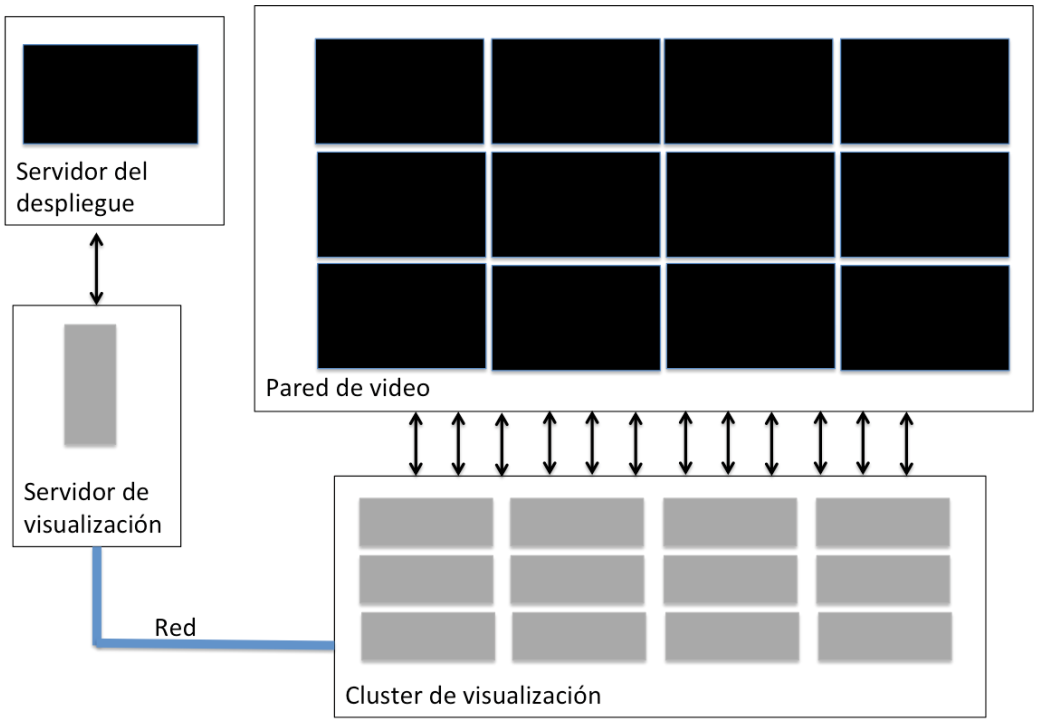


Figura 5.2: Esquema de una pared de vídeo controlada por un servidor de visualización.

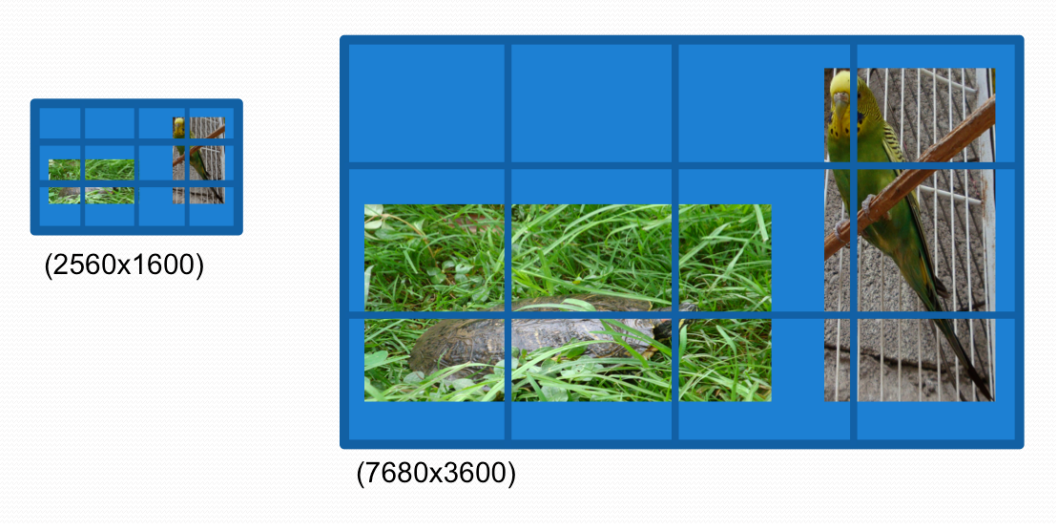


Figura 5.3: Funcionalidad del visor de imágenes

Se pueden admitir nuevos eventos siempre que sean anexados al mapa de eventos, donde se hace el reconocimiento de la acción solicitada. Los eventos básicos del visor de imágenes

son el arrastre del ratón con el botón derecho (para mover la imagen en la pantalla), y el click del botón derecho del mouse (para seleccionar la imagen).

El sistema puede definirse mediante un diagrama de estados (ver figura 5.4). Tenemos 6 estados por los que puede pasar el visor de imágenes; (1) estado inicial *sa*, (2) estado esperando *sb*, (3) estado consumidor *sc*, (4) estado despliegue *sd*, (5) estado acción *se* y (6) estado búsqueda *sf*. Cada estado es modificado mediante la generación de un evento ya sea automático o de usuario. Los eventos que pueden ocurrir son 7 y se nombran a continuación; (1) evento comenzar *e0*, (2) evento abrir *e1*, (3) evento desplegar *e2*, (4) evento recibir *e3*, (5) evento reconocer *e4*, (6) evento buscar *e5*, (7) evento comunicar *e6*

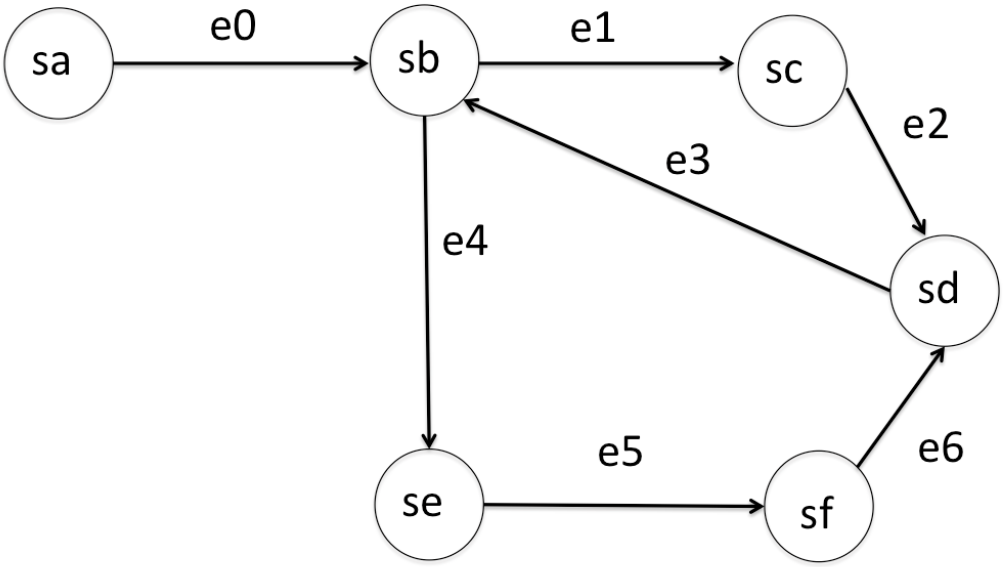


Figura 5.4: Grafo de estados del visor de imágenes

El estado inicial es *sa* en este estado el sistema esta apagado y requiere un evento *e0* para iniciar el proceso. El estado *sb* inicia cuando el sistema esta listo para abrir la primera imagen. En esta etapa la interfaz de usuario se desplegó en el servidor de visualización y las pantallas. Cuando el evento *e1* ocurre el sistema pasa al estado *sc*. A continuación se genera el evento *e2* para generar y enviar la imagen en las pantallas del cluster y pasa al estado *sd*. El evento *e3* se genera para obtener la posición dentro de la pantalla para desplegar y regresar al estado *sb*.

Cuando el sistema está en estado *sb* y alguna imagen fue desplegada con anterioridad, el sistema puede recibir eventos de usuario *e4*. Este evento cambia al sistema y lo pone en estado *se* en este estado el evento debe ser reconocido. Después se envían mensajes al cluster de visualización con la información de lo ocurrido *e5*. Luego el sistema pasa al estado de búsqueda *sf* para localizar el objeto que recibió el evento y enviar la información a los nodos del cluster. Lo siguiente es pasar al estado *sd* y generar el evento *e3* para

llegar al estado esperado *sb*.

El sistema proporciona dos tareas necesarias para abrir imágenes y para recibir eventos de usuario dentro del área de despliegue. Para lograr estas tareas el sistema pasa por diferentes estados, cada estado tiene sus propios requisitos. El estado inicial *sa* requiere conexiones de red entre el cluster de visualización y el servidor de visualizaciones. El estado de espera *sb* necesita que el servidor de visualización este iniciado y conectado, con la interfaz de usuario desplegada. Además se requiere la asignación de arreglos contenedores de objetos del lado del servidor y del cliente. El estado consumidor *sc* necesita enviar los mensajes con los datos y el control del servidor al cluster de visualización. También se encarga de almacenar los objetos generados y sus copias en el cluster. El estado desplegado *sd* necesita tomar los datos para generar la imagen en la pantalla, requiere conocer las posición y el tipo de imagen a desplegar. También necesita enviar la información al cluster de visualización. El estado acción *se* necesita hacer una búsqueda del objeto que generó el evento dentro del arreglo del servidor y del cluster para replicar la acción. También necesita saber el tipo de evento ocurrido mediante la información obtenida de la cola de eventos del sistema operativo. El estado de búsqueda *sf* espera el tiempo necesario para la localización remota del objeto para posteriormente pasar al estado de despliegue. Se forma un ciclo en el sistema que inicia y termina en el estado de espera *sb*.

A continuación se muestran los modelos de diseño del visor de imágenes con la metodología OMT, para explicar los requerimientos y funcionalidades del sistema.

## 5.2. Metodología OMT aplicada en el visor de imágenes

Dadas las necesidades básicas de un visor de imágenes que funcione sobre una pared de vídeo, se obtiene el siguiente modelo de requisitos:

- Se desea desarrollar un visor de imágenes que funcione sobre una pared de vídeo.
- Se reciben eventos de usuario sobre la GUI principal.
- Se muestra la GUI sobre el servidor de visualización y se replica sobre la pared de vídeo.
- Se utiliza un archivo de configuración inicial.
- Se muestran dos botones para permitir abrir las imágenes y configurar el aspecto de las nuevas imágenes.

Se plantea un conjunto de objetivos por alcanzar en el visor de imágenes. Para eso se define el modelo de objetivos (ver figura 5.5). Dentro de las principales objetivos que se tiene en este diseño es el manejo de la pared de vídeo y el manejo del objeto de despliegue que en este caso son imágenes. En este prototipo se limita el tipo de manejo para facilitar su uso a los usuarios menos experimentados. Se deja libre el manejo de la imagen con la finalidad de agregar fácilmente la aplicación de filtros mediante la interfaz

de configuración. La extensión de esta aplicación se puede logar mediante la adición de nuevos métodos al los objetos visuales.

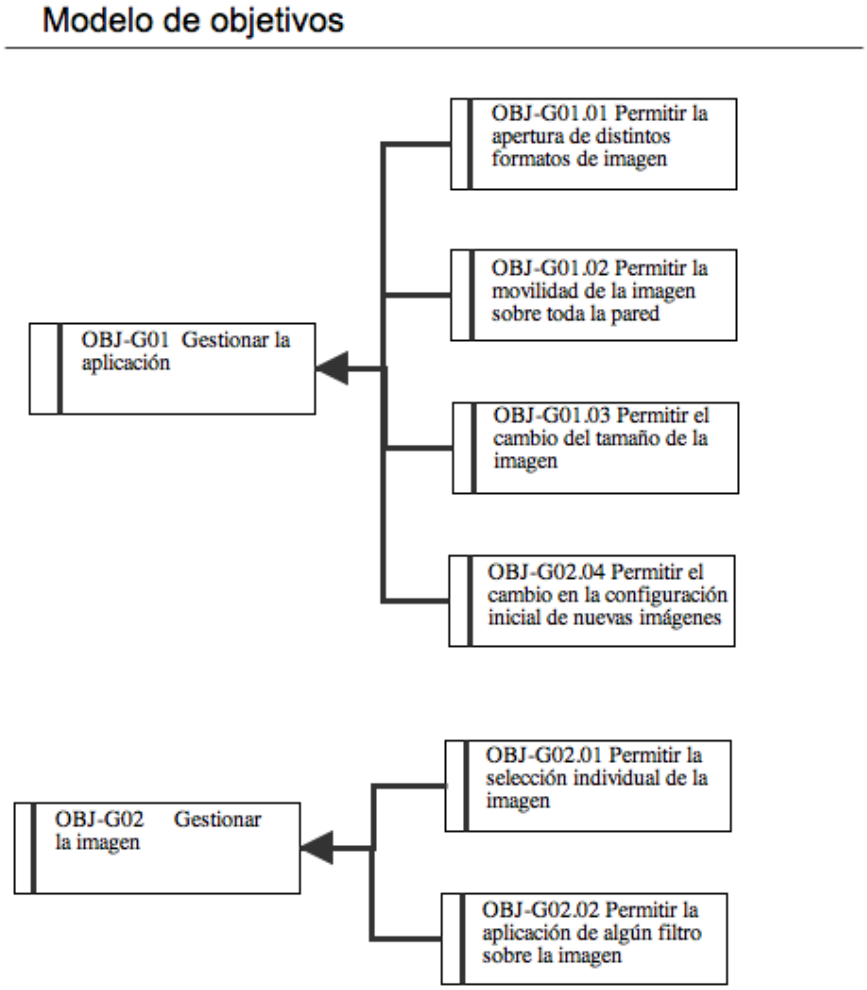


Figura 5.5: Objetivos de la aplicación

Un aspecto importante en la definición de aplicaciones de este tipo es la descripción de los requisitos de almacenamiento (ver figura 5.6). Este almacenamiento esta definido por el tipo de dato de entrada que en este caso son imágenes con los formatos más comunes (png, jpg, bmp, tiff). Además se mantiene información en un archivo de configuración que contiene la configuración visual y de red de la pared de vídeo (ip, puertos, resolución, posición matricial de las pantallas, etc). El almacenamiento estará definido dentro de cada nodo y en el servidor de visualización. La estrategia default será la replicación de los datos visualizados; con la finalidad de permitir su disponibilidad.

En este prototipo se permite el acceso a cualquier tipo de usuario, ya que las posibilidades de configuración estarán limitadas. El único actor que aparece en los siguientes

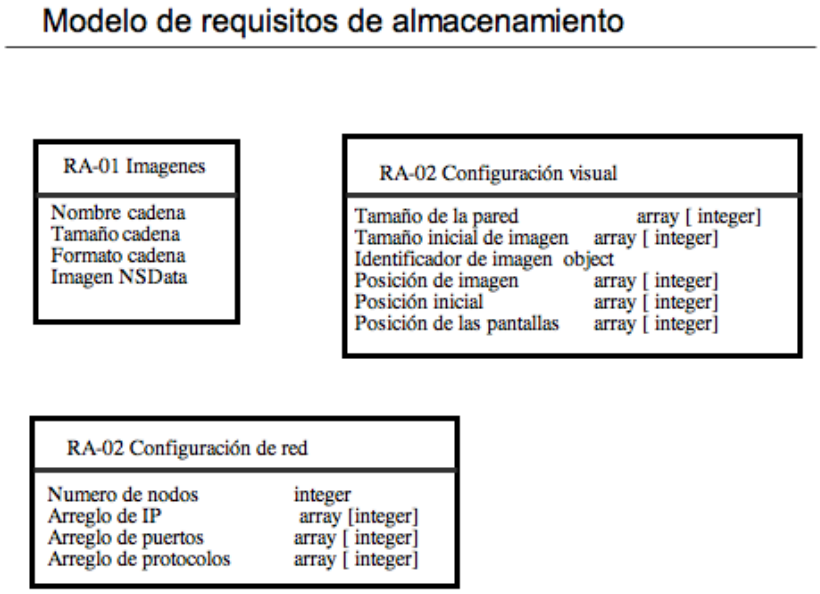


Figura 5.6: Requerimientos de almacenamiento de datos del visor

casos de uso es un usuario (ver figura 5.7).

| Identificador | AC-01 Usuario |
|---|---|
| Descripción | La aplicación debe prever el trato con los usuarios, que son las personas que pueden usar el visor de imágenes. |
| Hereda de | No |

Figura 5.7: Actor del sistema

Aquí se encuentran algunos escenarios que pueden ocurrir, es decir se muestran algunos ejemplos de como funciona el sistema. El enfoque usado para la realización de la especificación del sistema esta basado en UML (Lenguaje Modelado Unificado), mediante modelos gráficos como sistema de especificación. Se presentan los casos de uso para el usuario de la pared de vídeo, basados en una interfaz de usuario simple que permita la generación de un objeto visual y su manejo sobre la pared de vídeo. Tenemos tres servicios básicos necesarios; (1) servicio encargado de definir la configuración inicial del la pared de vídeo, incluye la información de los nodos, sus posiciones, IP, puertos y resolución, (2) servicio de control de objetos que permite la creación, eliminación y modificación de los objetos visuales y (3) servicio de manejo de eventos que se realiza de manera automática recibiendo la información desde el servidor de ventanas.

El primer caso de uso (ver figura 5.8) es utilizado cuando se abre una imagen, básicamente un usuario puede abrir su imagen como en cualquier aplicación de escritorio estándar.

| Nombre | UC-01 Abrir nueva imagen |
|---|---|
| Precondición | El sistema debe estar iniciado en estado esperando (sb). |
| Secuencia principal | 1.- El usuario da clic sobre el botón de abrir |
| | 2.- El sistema abre una caja de dialogo mostrando archivos |
| | 3.- El usuario selecciona la imagen deseada |
| | 4.- El sistema pasa al estado consumidor (sc) |
| | 5.- El sistema pasa al estado despliegue (sd) |
| | 6.- El sistema regresa al estado esperando (sb) |
| Alternativas/errores | 3.1.- Si sucede un error al abrir la imagen, el sistema muestra un mensaje de error y este caso de uso termina. |
| | 4.1.- Si sucede un error al enviar los mensajes al cluster se muestra un mensaje de error y este caso de uso termina. |
| | 5.1 Si sucede un error en el despliegue, el sistema muestra un mensaje de error y este caso de uso termina. |
| Post-condición | El nuevo objeto visual (imagen) se almacena en el arreglo del servidor y del cluster. |

Figura 5.8: Caso de uso: Abrir nueva imagen

El proceso se inicia con un botón encontrado en la interfaz de usuario ubicada en el servidor de visualizaciones. Se abre un cuadro de diálogo standard con el que se selecciona la imagen a desplegar (ambiente de cocoa en la plataforma Mac OSX). La conexión y envío de mensajes entre nodos y servidor son transparentes( ver figura 5.9).

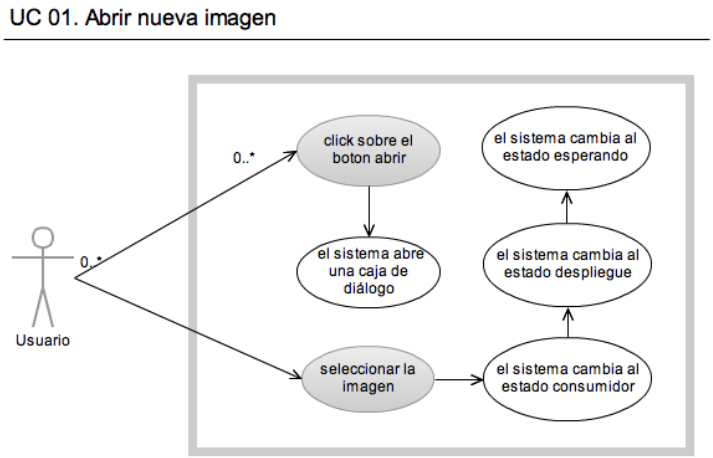


Figura 5.9: Diagrama de caso de uso: Abrir nueva imagen

El segundo caso de uso (ver figura 5.10) explica el comportamiento cuando se mueve una imagen desplegada, esto mediante el arrastre del ratón sobre la imagen seleccionada.

Esta es una especificación del manejo de eventos y es posible enviar solo la información de la posición actual para encontrar el objeto en el arreglo y la nueva posición para actualizar en la pared de vídeo. Esta ventaja se ha utilizado en el protocolo VNC, disminuyendo el control a dos enteros (actualizados durante el arrastre). Se debe considerar un retraso generado por el despliegue y borrado de la ventana conforme es trasladada.

| Nombre | UC-02 Mover una imagen |
|---|---|
| Precondición | Estado esperando (sb) y al menos una imagen desplegada con anterioridad. |
| Secuencia principal | 1.- El usuario posiciona el mouse sobre la imagen deseada |
| | 2.- El usuario arrastra la imagen y la cambia de posición |
| | 3.- El sistema recibe el evento lo reconoce y lo replica al cluster. |
| | 3.1.- El sistema pasa al estado acción (se). |
| | 3.2.- El sistema pasa al estado búsqueda (sf). |
| | 3.3.- El sistema pasa al estado despliegue (sd). |
| | 4.- El sistema regresa al estado esperando (sb). |
| Alternativas/errores | 3.1.1.- Si sucede un error en el reconocimiento del evento, el sistema muestra un mensaje de error y este caso de uso termina. |
| | 3.1.2.- Si sucede un error al enviar los mensajes al cluster se muestra un mensaje de error y este caso de uso termina. |
| | 3.3.1.- Si sucede un error en el despliegue, el sistema muestra un mensaje de error y este caso de uso termina. |
| Post-condición | El evento ocurrido en el despliegue del servidor es replicado en el despliegue del cluster de visualización. |

Figura 5.10: Caso de uso: Mover una imagen

El proceso inicia cuando el usuario posiciona el puntero del ratón en una imagen desplegada y empieza el arrastre sosteniendo el botón derecho del ratón.

Cuando el evento es reconocido se hace la búsqueda del objeto en el contenedor y se envía el mensaje a los nodos. Los nodos localizan su propio objeto he inician la actualización de la posición de la imagen en la pared de vídeo. La búsqueda se realiza una sola ocasión al inicio del reconocimiento del evento. Después se envían solo las coordenadas globales[1] actuales de la imagen (ver figura 5.11).

El tercer caso de uso es también una especificación del manejo de eventos, en este caso basta seleccionar la imagen y emplear una combinación de teclas para aplicar un filtro sobre la imagen seleccionada.

En este caso se puede mantener una lista con los filtros disponibles de modo que el mensaje enviado desde el servidor de visualizaciones sea solo un entero (ver figura 5.12).

[1]Las coordenadas globales son la posición(x,y) que ocupa la imagen con respecto al despliegue total de la pared de vídeo.

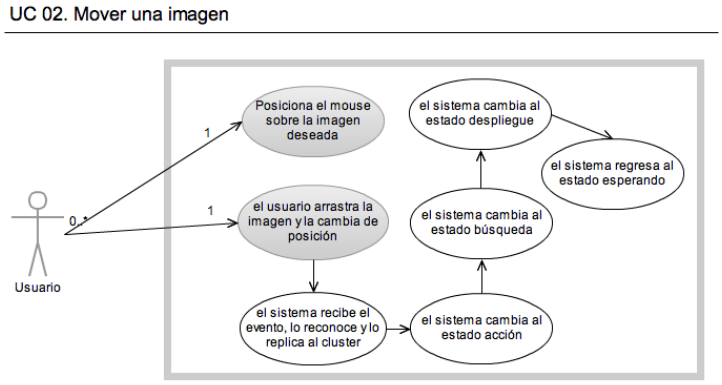


Figura 5.11: Diagrama de caso de uso: Mover una imagen

| Nombre | UC-03 Aplicar filtro a una imagen |
|---|---|
| Precondición | Estado esperando (sb) y al menos una imagen desplegada con anterioridad. |
| Secuencia principal | 1.- El usuario posiciona el mouse sobre la imagen deseada |
| | 2.- El usuario da click derecho sobre la imagen para seleccionarla |
| | 3.- El usuario selecciona el filtro y lo aplica con la combinación de teclas |
| | 4.- El sistema recibe el evento lo reconoce y lo replica al cluster |
| | 4.1.- El sistema pasa al estado acción (se). |
| | 4.2.- El sistema pasa al estado búsqueda (sf). |
| | 4.3.- El sistema pasa al estado despliegue (sd). |
| | 5.- El sistema regresa al estado esperando (sb). |
| Alternativas/ errores | 4.1.1 Si sucede un error en el reconocimiento del evento, el sistema muestra un mensaje de error y este caso de uso termina. |
| | 4.1.2 Si sucede un error al enviar los mensajes al cluster se muestra un mensaje de error y este caso de uso termina. |
| | 4.3.1 Si sucede un error en el despliegue, el sistema muestra un mensaje de error y este caso de uso termina. |
| Post-condición | El evento ocurrido en el despliegue del servidor es replicado en el despliegue del cluster de visualización. |

Figura 5.12: Caso de uso: Aplicar filtro a una imagen

El proceso inicia con la selección de la imagen, en general esta selección es necesaria para permitir que los nodos busquen la imagen que será afectada. De ese modo es posible realizar la aplicación del filtro en toda la pared casi simultáneamente[2] (ver figura 5.13).

El ultimo caso de uso (ver figura 5.14), explica como un usuario puede modificar algunos aspectos de configuración durante el tiempo de ejecución. Este tipo de configuración es

[2]Consideramos el retraso del envío de mensaje y la sincronización entre los nodos y el servidor mínimo, el mayor tiempo se gasta en la aplicación del filtro en las imágenes dependiendo de su tamaño.

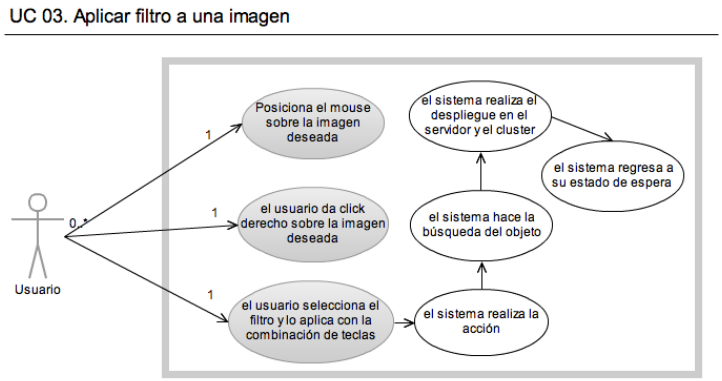


Figura 5.13: Diagrama de caso de uso: Aplicar filtro a una imagen

proporcionada para modificar el aspecto inicial de una imagen al ser desplegada (tamaño, color, filtros, posición inicial). Esto nos permite el manejo transparente de la aplicación dejando los aspectos de redes y despliegue en la pared en otro nivel.

| Nombre | UC-04 Modificar la configuración inicial |
|---|---|
| Precondición | Estado esperando (sb) |
| Secuencia principal | 1.- El usuario da clic sobre el botón de configuración |
| | 2.- El sistema abre una caja de dialogo mostrando la configuración inicial |
| | 3.- El usuario selecciona la configuración deseada |
| | 4.- El sistema actualiza la configuración y regresa al estado esperando (sb) |
| Alternativas/ errores | 5.1.- Si sucede un error en el despliegue, el sistema muestra un mensaje de error y este caso de uso termina. |
| Post-condición | La configuración inicial es actualizada con los nuevos cambios. |

Figura 5.14: Caso de uso: Modificar la configuración inicial

La sucesión de pasos en este caso de uso son los siguientes: (1) el usuario da click sobre un botón encontrado en la interfaz principal, (2) se abre un cuadro de diálogo que permite seleccionar diversas especificaciones referentes al despliegue de la imagen y (3) se seleccionan los cambios y se guarda la configuración. A partir de ese momento el usuario podrá trabajar con la nueva configuración (ver figura 5.15).

El prototipo de visualización obtenido del diseño anterior (ver figura 5.16) muestra los componentes de la aplicación completa y su relación.

Las frases se relaciona con la funcionalidad asociada de la siguiente manera FR01 esta relacionada con UC-01, FR02, FR03 se relacionan con UC-03, FR02 y FR04 se relacionan con UC-02 y FR05 se relaciona con UC-04.

El diagrama de generado muestra los estados por los que se tiene que pasar para lograr las funcionalidades previamente descritas (ver figura 6.3).

Con anterioridad se mencionó el funcionamiento de los estados por los que tiene que pasar la aplicación. La importancia de mantener los estados esta dada por la necesidad

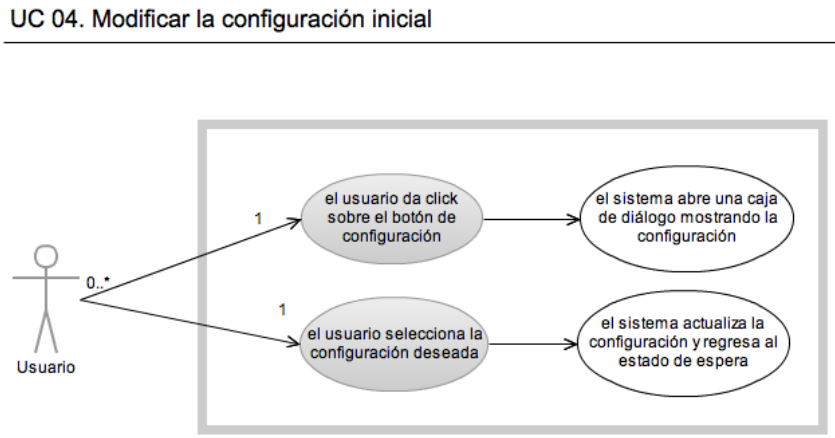


Figura 5.15: Diagrama de caso de uso: Modificar la configuración inicial

| Nombre | PV01 Visor de imágenes |
|---|---|
| Actores | AC-01 Usuario |
| Descripción | El sistema debe permitir la visualización de las imágenes y la navegación expresada y que representa los enlaces del sistema. |
| Frases | FR01 Apertura de imagen en la pantalla |
| | FR02 Selección de la imagen |
| | FR03 Aplicación de filtro en la imagen |
| | FR04 Cambio de posición de la imagen en la pantalla |
| | FR05 Cambio de datos de configuración |
| Funcionalidad asociada | UC-01 Abrir nueva imagen |
| | UC-02 Mover imagen |
| | UC-03 Aplicar filtro a una imagen |
| | UC-04 Modificar la configuración inicial |
| Información visualizada | RA01. Imagen |

Figura 5.16: Prototipo de visualización PV-01. Visor de imágenes

de enviar información entre el servidor y los nodos. Manteniendo este orden podemos permitir el reconocimiento remoto de los cambios ocurridos en la interfaz de usuario.

Podemos definir dos diagramas de secuencia que pueden ocurrir. El primero es cuando el sistema es iniciado. El segundo ocurre cuando se genera un nuevo objeto visual o se genera un nuevo evento.

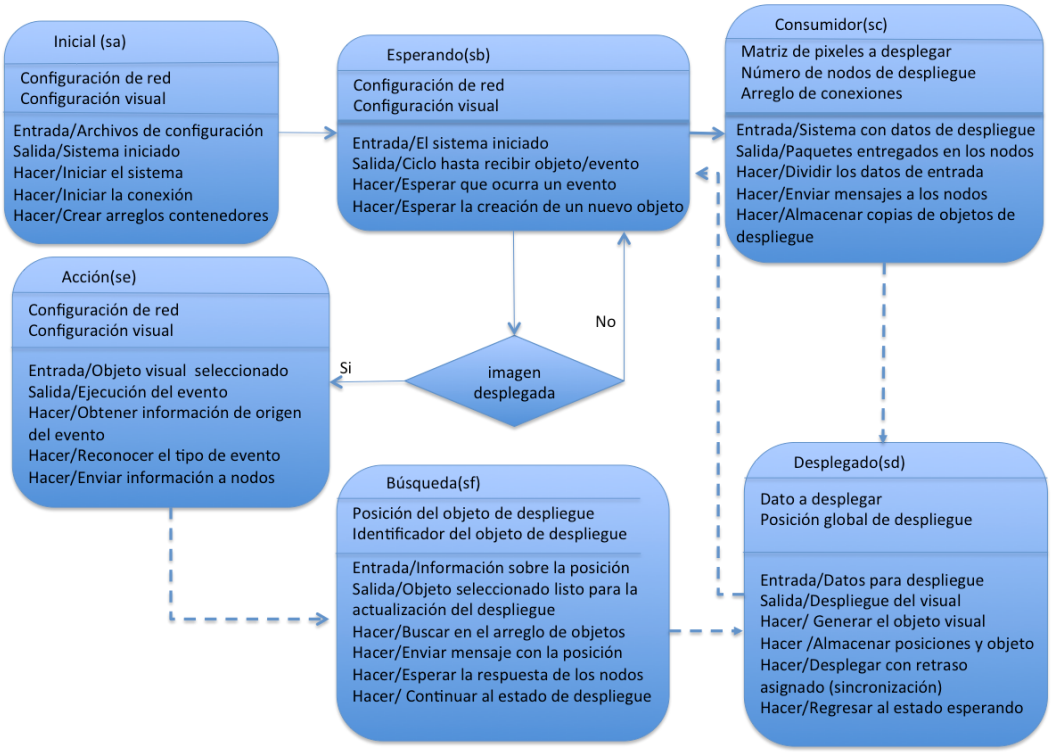


Figura 5.17: Diagrama de estados DE-01. Visor de imágenes

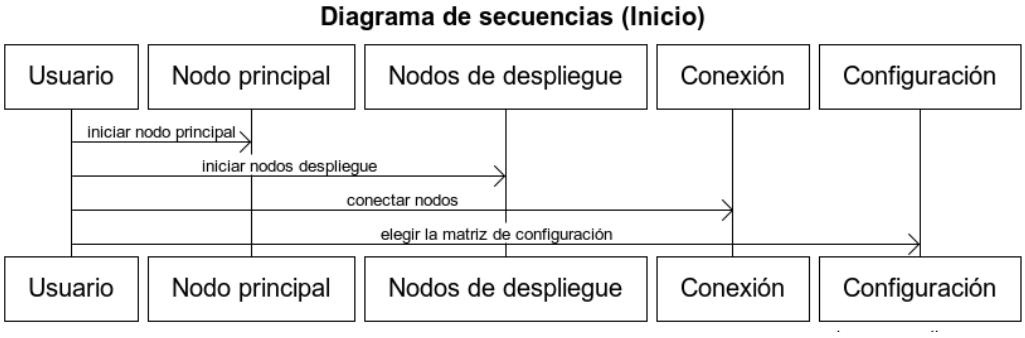


Figura 5.18: Diagrama de secuencia DS01. Visor de imágenes

Cuando el sistema inicia el primero que recibe la acción es el nodo principal. Este nodo debe iniciar a los nodos despliegue. Posteriormente el nodo principal inicial la conexión con los nodos despliegue. Después de iniciados emplean el archivo de configuración para conocer las características de la visualización (ver figura 5.18).

Cuando el sistema esta iniciado puede recibir eventos de usuario o la creación de un nuevo objeto visual. Cuando el objeto es agregado al nodo principal se envían los datos

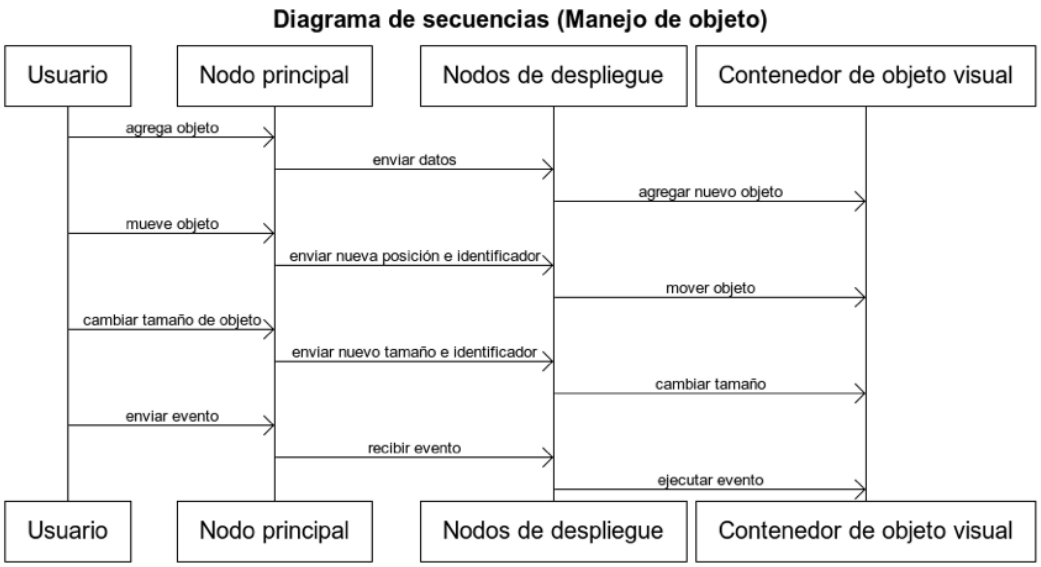


Figura 5.19: Diagrama de secuencia DS02. Visor de imágenes

a los nodos de despliegue. Cada nodo agrega a su objeto visual dentro de su objeto contenedor. Cuando un evento ocurre en el nodo principal la información es encapsulada y notificada a los nodos despliegue (ver figura 5.19).

## 5.3. Diagrama de clases

En esta sección se muestran las clases definidas para cada capa del modelo DVO ( ver figura 5.20 ). Cada objeto se comunica directamente con los objetos en las capas superiores e inferiores.

A continuación se explican las clases básicas del visor de imágenes. Algunas de estas clases están definidas empleando patrones de diseño. La implementación de estas clases permite el funcionamiento de la capa DVO.

Las clases que definirán los objetos del cliente y del servidor son similares. De modo que se explicará cuando existan diferencias para cada caso. Por ejemplo, el objeto de la clase control es el encargado de obtener la configuración inicial y se encuentra tanto en el cliente como en los nodos. El objeto control del lado del cliente inicia la conexión con los nodos y se emplea un objeto de clase conexión. Al obtener la información de configuración se crea un objeto de tipo diccionario. Cuando el sistema se inicia normalmente se generan los objetos aplicación y contenedor del lado del cliente y de los nodos. El cambio ocurre al recibir la información. En esta etapa el cliente y los nodos realizan casi las mismas acciones. La diferencia es que el cliente inicia la conexión y los nodos publican sus servicios.

Cuando se termina la inicialización se espera recibir una acción mediante un objeto de control tipo botón ubicado en la interfaz principal del cliente. Cuando se accione

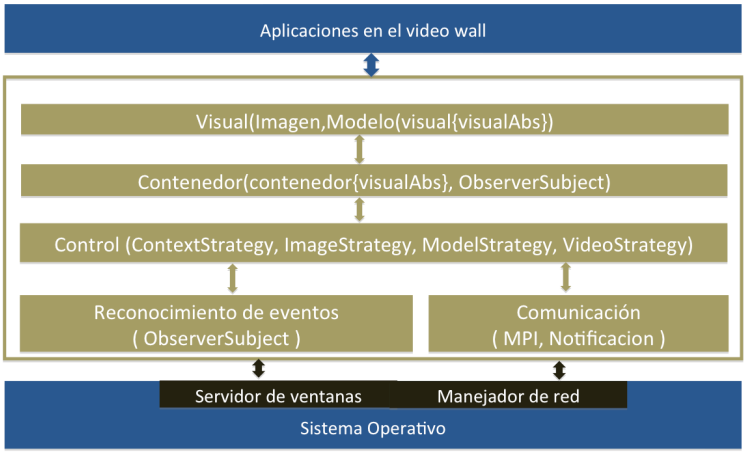


Figura 5.20: Clases definidas para cada capa del modelo DVO

este objeto se hace un llamado al objeto de la clase distribución y al objeto de la clase despliegue. Estos objetos se encargan de obtener y dividir los datos. Posteriormente se genera el objeto visual en la interfaz principal. Mediante el objeto conexión se envía una notificación a los nodos con el dato generado. Este dato depende del modo de despliegue, del número de nodos y de su posición en el despliegue final. Cuando el cliente recibe un evento desde su interfaz principal, los datos del evento son obtenidos desde el servidor de ventanas del sistema operativo local. Después son reconocidos por el objeto de la clase evento (el cual debe hacer una búsqueda en el diccionario de eventos), que regresará la instrucción al objeto control para que este se encargue de enviar el mensaje a los nodos.

Cuando se generar un nuevo objeto visual, los nodos reciben la petición desde el objeto control. Este objeto envía los datos al contenedor para generar el objeto despliegue. El contenedor se encarga de crear el nuevo objeto, desplegarlo y almacenarlo en un arreglo. En el caso de recibir un mensaje de evento el contenedor debe buscar el objeto que lo generó dentro del cliente. Después se envía la información para la búsqueda del objeto en los nodos. Cuando se localizan los objetos se realiza la acción definida para el tipo de evento ocurrido.

Se deben considerar algunos aspectos relevantes para la implementación de los patrones de diseño. Estos aspectos involucran las decisiones finales para las definiciones de las clases que conformarán la implementación del modelo DVO.

El modelo DVO esta definido como una capa de objetos distribuidos visuales, cuya finalidad principal es definir un componente que permita el desarrollo de un manejador de ventanas distribuido. Existe un objeto View definido con anterioridad y basado en el patrón MVC, este objeto es la base de algunos manejadores de ventanas ya que define la vista de las aplicaciones. Este objeto View es tomado y definido para nuestros propósitos como objeto Visual, para proporcionarle una definición separada del patrón. En esta ocasión la definición de esta visualización es tomada más como un objeto independiente de manera que su distribución permita la creación de la capa DVO. Aprovechamos las

ventajas obtenidas de las definiciones anteriores y aplicamos nuevas estrategias a fin de lograr la mejor definición del modelo propuesto.

### 5.3.1. Implementación del patrón Composite (Capas contenedor e interfaz visual)

La definición de este patrón de diseño fue presentada en capítulos anteriores, en esta sección toca mencionar aspectos necesarios para la implementación. Este patrón permite la definición de clases jerárquicas consistentes en objetos primitivos y objetos compuestos. Los objetos primitivos pueden ser compuestos en objetos complejos, los cuales pueden ser compuestos y así recursivamente. Cualquier código de cliente que sea un objeto primitivo puede ser agregado a un objeto compuesto.

Para nuestros propósitos definimos los objetos primitivos como una imagen, un vídeo o un modelo 3D. El objeto visual puede contener alguno de los objetos primitivos. El objeto visual puede contener a otro objeto visual, en ese caso el objeto visual se convierte en contenedor. Otros objetos primitivos pueden ser agregados posteriormente. Para realizar esto necesitamos emplear una clase abstracta que permita heredar a las clases visual y contenedor (ver figura 5.21).

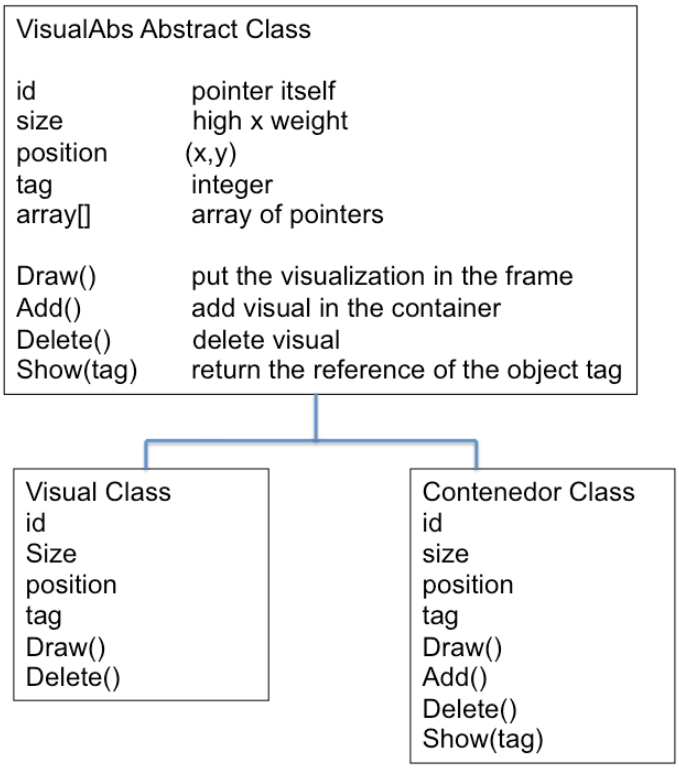


Figura 5.21: Clase abstracta

El control es el encargado de comunicarse con el contenedor y con los visuales, por lo que debemos especificar una referencia al objeto visual o al compuesto de objetos para su manipulación. La referencia debe ser estandarizada para ambos la llamaremos ID. El ID de un objeto visual apunta a su dirección. El ID de un compuesto debe apuntar al objeto contenedor. De esta manera el manejo del control se simplifica. El objeto contenedor debe encargarse posteriormente de modificar a sus objetos internos o compuestos, de manera recursiva.

Creando subclases de la clase abstracta podemos definir una subclase contenedor que almacene instancias de la subclase visual. Con una operación ADD que reciba la referencia ID de un objeto visual. De esta forma podemos agregar objetos visuales automáticamente con estructuras existentes.

No se van a contemplar restricciones para los componentes en la composición debido a que es una versión preliminar con componentes básicos. Pero cuando los componentes incrementen las restricciones deben ser impuestas.

Se emplean referencias explícitas para el manejo de los objetos visuales desde el contenedor. Mediante estas referencias ID, es posible facilitar el manejo de las estructuras y la eliminación de componentes. La operación DELETE con el ID del objeto elimina un objeto visual. Para la eliminación de contenedores, se debe hacer la eliminación de la descendencia completa. Es el contenedor el encargado de eliminar a sus componentes internos cuando se inicie su eliminación y la referencia en el contenedor superior debe ser eliminada.

No se contempla compatir componentes. El enfoque principal tratara de distribuir los componentes dentro de los nodos o replicar los datos contenidos en los visuales. De manera que ningún objeto visual puede tener más de un contenedor a la vez.

La clase abstracta de la cual derivan la subclase visual y la subclase contenedor tienen las operaciones básicas para el funcionamiento de ambas. De manera que el control pueda emplear su referencia ID y las trate de la misma manera. La clase abstracta proporciona implementaciones default para estas operaciones. Las clases componentes y visual serán sobrescritas posteriormente.

El principio de diseño de la clase jerárquica dice que una clase debe definir solo operaciones que son conocidas en sus subclases. De modo que las operaciones de la clase abstracta deben ser empleadas en la subclase visual y la subclase contenedor. En este aspecto las operaciones ADD y DELETE deben ser declaradas en la clase abstracta. De esta manera todos los componentes podrán ser tratados uniformemente (ver figura5.22).

Para mantener un orden al agregar un hijo se debe incrementar un contador almacenado en un TAG como un entero, de esta manera tendremos el arreglo con los objetos visuales ARRAY. Cada objeto visual debe mantener el valor de su TAG. Este valor es incremental y los identifica con respecto al tiempo en el que se agregó. Cuando un objeto visual es eliminado de un contenedor el arreglo se reduce y nuevos TAG deben ser asignados.

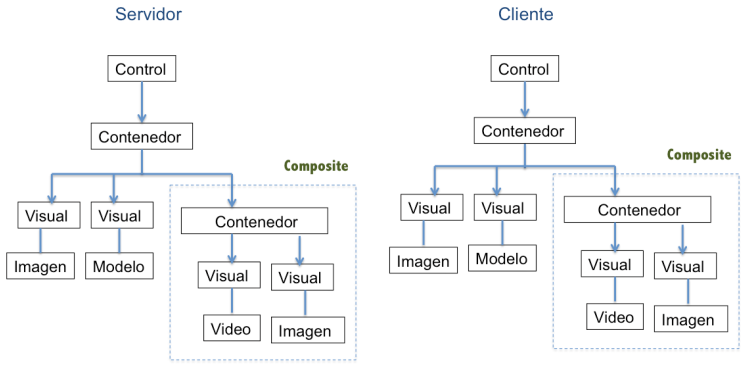


Figura 5.22: Patrón Composite

### 5.3.2. Implementación del patrón Observer (Capas de reconocimiento de eventos y contenedor)

En este modelo emplearemos el patrón Observer en modo local y remoto. Como se mencionó con anterioridad este patrón nos permite mantener una observación de lo que ocurre con un objeto en particular. En el caso local debemos mantener el control de los objetos visuales, para notificar los cambios que ocurran en cada objeto visual. Con el patrón Composite lograremos manejar a los objetos visuales y a sus contenedores del mismo modo. Con el patrón Observer podemos enterar a los visuales de los cambios hechos en sus contenedores. Para esto necesitamos tener dos clases auxiliares que nos permitan este manejo. (1) La clase Control es la que debe saber lo que ocurra en las visualizaciones. (2) La clase Observador será quien mantenga un ARRAYSUBJECTS donde se almacenarán las referencias de los objetivos que en este caso son los visuales y los contenedores. Además de mantener las referencias debe mantener un ARRAYSTATES con los valores de los estados actuales. Cuando algún cambio de estado ocurra debe cambiar el valor de su arreglo e informar al control el cambio. En este caso mantendremos una lista de estados para que el control sepa cual fue el cambio ocurrido.

En el caso remoto los nodos son los observadores y el objetivo es el control del cliente. De esta manera los nodos podrán mantener actualizadas sus visualizaciones mediante notificaciones que indiquen el cambio realizado. Adicionalmente se agregan argumentos del tipo de cambio para disminuir en número de mensajes enviados (ver figura 5.23).

Algunos de los problemas generados por este patrón están ligados a la dependencia generada entre los observadores y los objetivos. Cuando mapeamos objetivos en los observadores, podemos habilitar una operación que notifique los cambios y almacene referencias explícitas de los objetivos. Mantener un arreglo con los estados y las referencias de los objetos permitirá que los observadores sepan que objeto cambio y cual es su nuevo estado. Agregando argumentos de los cambios ocurridos podemos replicar el cambio en los observadores.

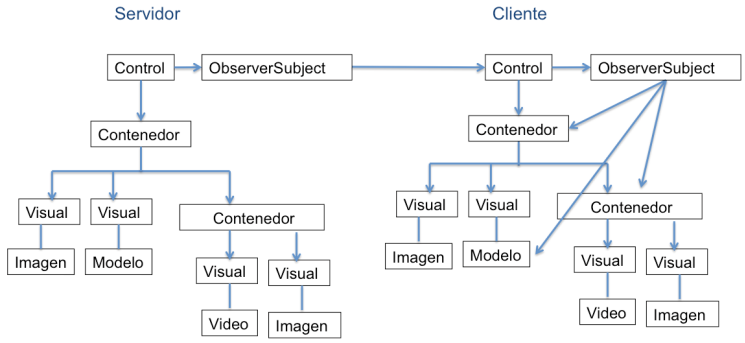


Figura 5.23: Patrón Observer

La notificación será enviada desde el objetivo cuando su estado cambie, dependiendo del cambio ocurrido se envían argumentos para permitir su replicación. Cuando un objetivo es eliminado, debe informar a sus observadores, de modo que puedan eliminarlo de sus arreglos de estado y de objetivo.

Un este caso se combinará la clase del observador con los objetivos de modo que de las clases mencionadas anteriormente solo se agregará una extra llamada ObserverSubject (ver figura 5.24).

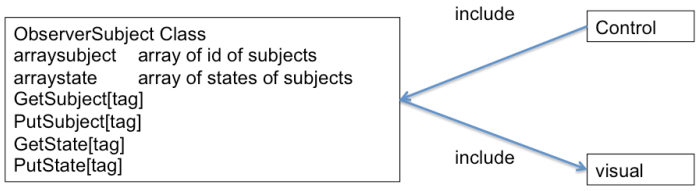


Figura 5.24: Clase Observer-Subject

### 5.3.3. Implementación del patrón State (Capa de comunicación)

El patrón de diseño State permite a un objeto alterar su comportamiento cuando ocurran cambios internos de estado. Este tipo de comportamiento lo tendremos al realizar la conexión de los nodos y los clientes. Mantendremos una comunicación híbrida usando MPI para enviar los datos iniciales y notificaciones provenientes de los observadores antes mencionados, para informar los cambios en los objetos visuales del cliente. Para lo cual requerimos la clase Comunicación. Esta clase se encargará de manejar el estado proporcionado por los observadores, dependiendo del estado y de la acción por la que los objetivos hayan pasado se elige el tipo de comunicación.

Los estados de un objeto visual simple son tres: start, display, wait. Para llegar a cada uno de ellos se genero alguna de las siguientes acciones:

- Del estado Start al estado despliegue se tienen que enviar los datos a desplegar.
- Del estado Display al estado wait se pasa inmediatamente después de terminar el despliegue.
- Del estado Wait se puede ir al estado Start cuando se genere un nuevo objeto visual o nuevos datos para despliegue o al estado Display si es que se requiere una actualización de tamaño, posición o control sin requerir cambio de datos.

Por lo que los observadores deben enviar no solo el estado si no la acción que los hizo cambiar; para poder llegar al mismo estado. Dependiendo de eso la conexión debe elegir el tipo de mensaje enviado con la clase Notification o con la clase MPI.

La clase Comunication es la encargada de mantener la información de la conexión IP, PUERTO y el SOCKET que se utilizará. Esta clase estará en el cliente y el servidor con sus respectivos cambios (ver figura 5.25).

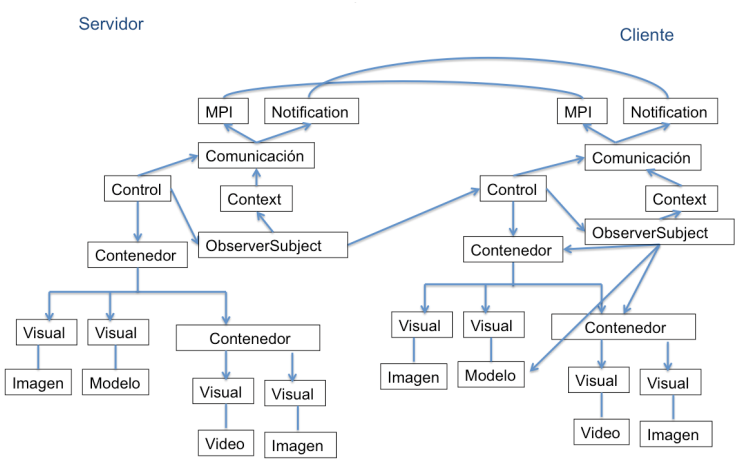


Figura 5.25: Patrón State

Context es la clase que mantendrá los cambios en un arreglo, estos cambios son obtenidos por las instancias de los observer. La instancia de Context es empleada en la clase Comunication quien empleará ya sea su objeto MPI o su notificación dependiendo la información de Context. El objeto ObserverSubject es el encargado de definir la transición del estado, de modo que el objeto Context sea fijo (ver figura 5.26).

### 5.3.4. Implementación del patrón Strategy ( Capa de control )

El patrón Strategy permite usar diferentes algoritmos dependiendo del contexto, en nuestro caso los 3 tipos de datos que empleamos imagen, modelo y vídeo, requieren de diferentes algoritmos para su despliegue y distribución.

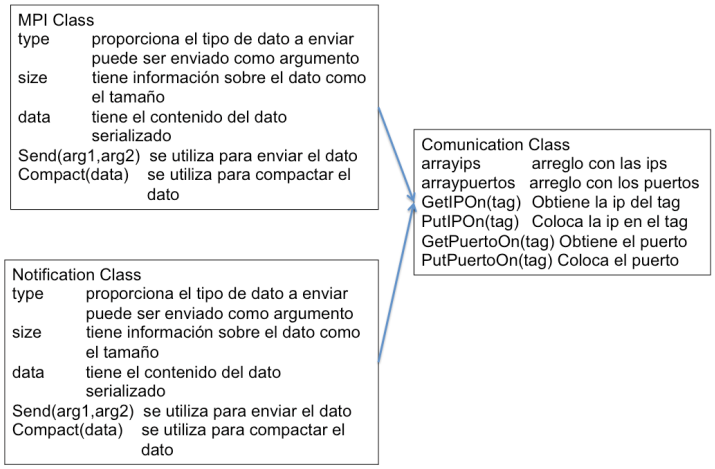


Figura 5.26: Clases State

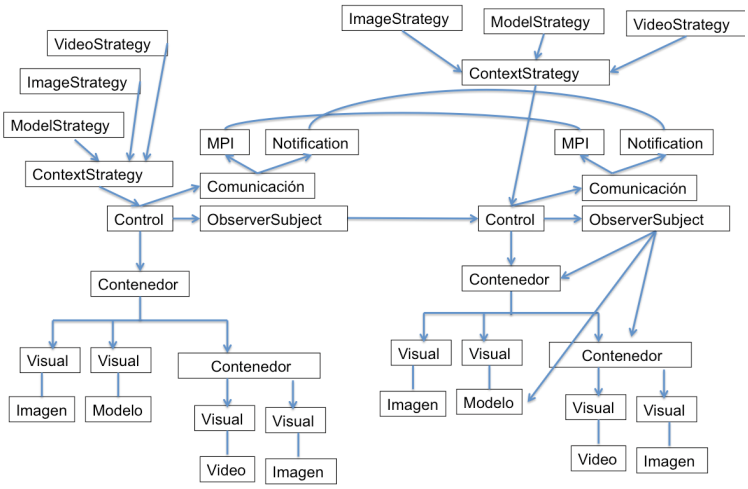


Figura 5.27: Patrón Strategy

La distinción de entre los 3 tipos se hace mediante su extensión, de modo que desde que se obtenga el nombre del archivo se puede hacer la elección del algoritmo. Para eso se agrega una clase ContextStrategy que se encarga de definir el tipo de algoritmo necesario dependiendo de la extensión del archivo. Cada tipo debe definirse con una clase de modo que tenemos 3 clases vídeoStrategy, ImageStrategy y ModelStrategy (ver figura 5.27).

En este caso la implementación de cada algoritmo tiene como argumento la referencia a los datos, que dependiendo de su extensión deben ser tratados en diferentes formas. Adicionalmente en esta parte podemos combinar la distribución de los datos informando el cambio de estado a los ObserverSubject. De esta manera podemos aprovechar el algoritmo

de vídeoStrategy, ImageStrategy y ModelStrategy. No solo para abrir el dato sino además para distribuirlo (ver figura 5.28).

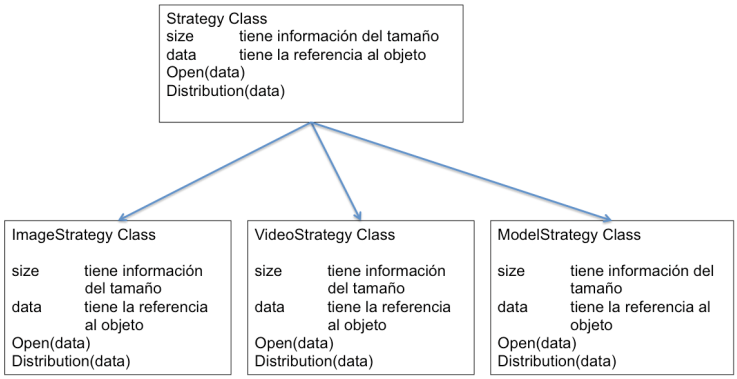


Figura 5.28: Clases Strategy

## 5.4. Pruebas de desempeño del visor de imágenes

El visor de imágenes fue implementado sobre el CinvesWall siguiendo el modelo DVO. Se implementó una estrategia básica de distribución de despliegue. Se realizaron múltiples pruebas con diversas imágenes. Todas ellas mostraron un comportamiento similar, se eligieron dos de las más representativas. A continuación se muestran las gráficas comparativas de la distribución del despliegue y el envío replicado.

La primera imagen tiene una resolución original de 24000x12000 pixeles. El tamaño del CinvesWall es de 7920x3600 pixeles. Al enviar la imagen replicada el tamaño del envio es de 24000x12000x32Mb= 8.583 Gb. Al enviar el paquete fraccionado el tamaño del envio es de 1920x1200x64 = 70.3 Mb. Además el tamaño de las fracciones permanece constante y el de la imagen original es variable.

Al realizar esta prueba se tomaron en cuenta los siguientes factores:

- La resolución de cada pantalla esta limitada por la tarjeta gráfica, por lo que el envío de mayor información de la que puede ser desplegada es innecesario.
- El tamaño de la imagen enviada impacta directamente en el desempeño de la comunicación y en la capacidad de respuesta del cluster de visualización.
- El tiempo que se ocupa en la distribución de la imagen y el envío de paquetes pequeños es más constante que el del envío de paquetes más grandes. Además los paquetes grandes con el tiempo decrementan el desempeño de la red.

Con la segunda imagen ocurre algo similar, el paquete completo es de 18001x11438x32 Mb= 6.136 Gb. Al enviar el paquete fraccionado el tamaño del envio es de 1920x1200x64 = 70.3 Mb (es constante).

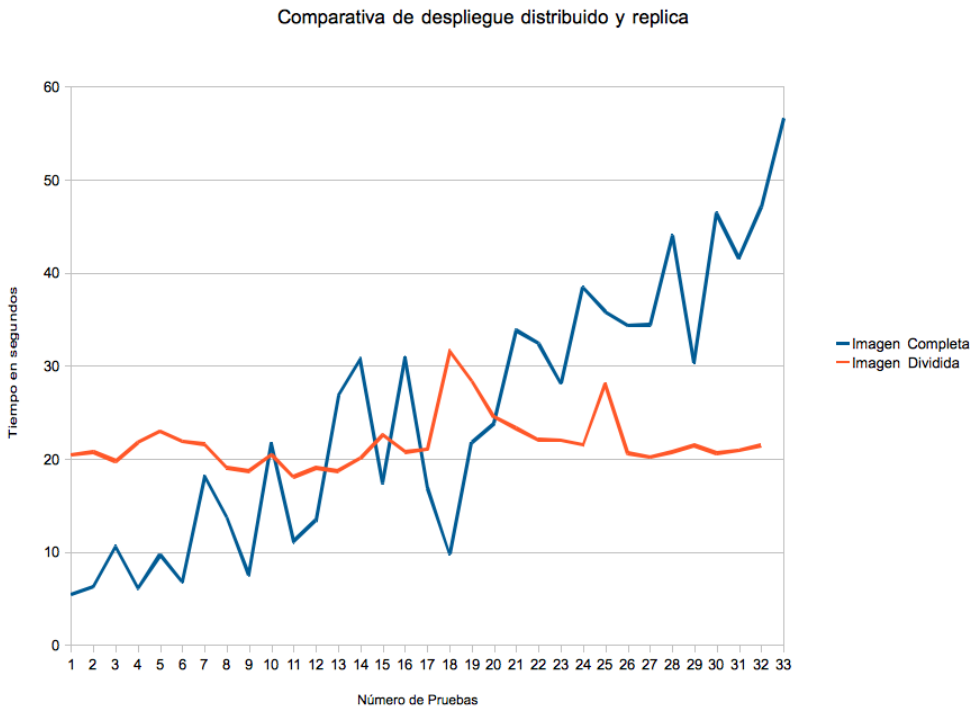


Figura 5.29: Imagen original de 24000x12000

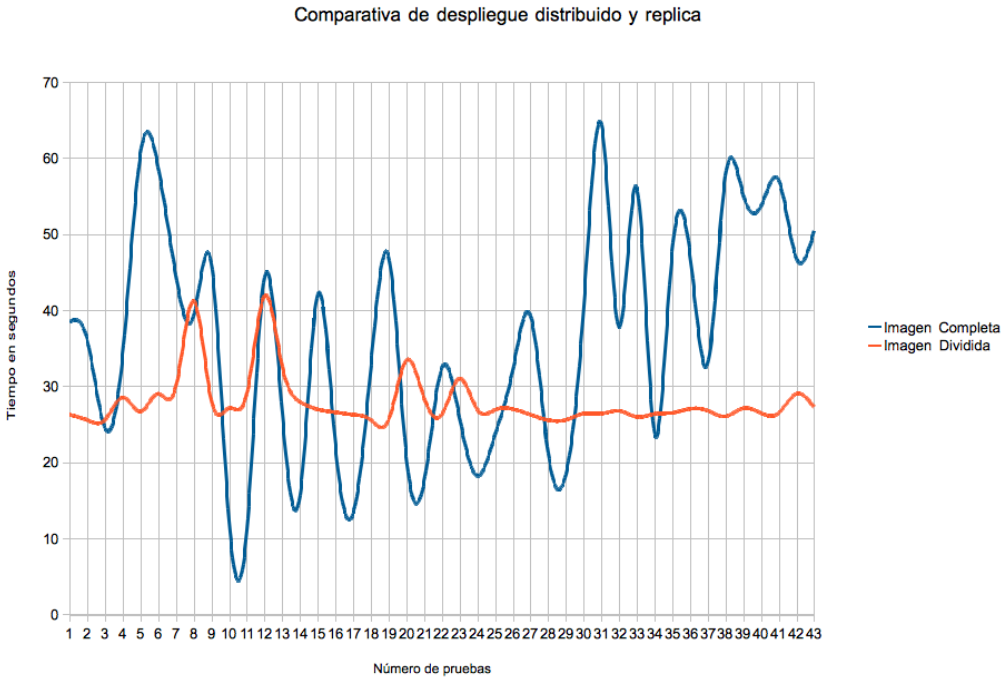


Figura 5.30: Imagen original de 18001x11438

Capítulo 6

# Conclusiones

## 6.1. Resultados

Durante este documento se mostró el análisis, diseño, desarrollo y uso del modelo de objetos distribuidos visuales. Este modelo sirve para permitir el manejo de paredes de vídeo. Es por ello que se describieron y discutieron algunos de los principales desarrollos que se emplean en este manejo, específicamente render paralelo y memoria compartida. Además se presentó el entorno de desarrollo necesario para el funcionamiento del modelo.

- Se clasificaron dos enfoques principales empleados para el manejo de paredes de vídeo:(1) render paralelo y (2) memoria compartida (capítulo 1).
- Se compararon y discutieron los enfoques de render paralelo y memoria compartida (capítulo 1).
- Se definió y describió el entorno de desarrollo involucrado en el control de paredes de vídeo (capítulo 2).
- Se hizo el análisis del los modelos de control de paredes de vídeo (capítulo 3).
- Se identificaron las funciones generales necesarias para el control de paredes de vídeo (capítulo 3).
- Se definió una abstracción jerárquica que permite definir el problema que implica el manejo de paredes de vídeo independientemente de la aplicación final y del hardware utilizado (capítulo 3).
- Se hizo el análisis de los modelos de control de paredes de vi?deo en el proceso de la distribución de despliegue (capítulo 3).
- Se hizo el análisis de los modelos de control de paredes de vídeo en el proceso de la distribución de datos (capítulo 3).

- Se hizo el análisis de los modelos de control de paredes de vídeo en el proceso de manejo de eventos (capítulo 3).
- Se diseño un modelo de control de paredes de vídeo que cumple con los procesos de distribución de datos, distribución de despliegue y manejo de eventos (capítulo 3).
- Se presentó la propuesta del modelo DVO basado en el modelo cliente-servidor, tomando como base las características de modelo MVC (capítulo 4).
- Se muestran y describen las API's propuestas para cada capa de modelo DVO (capítulo 4).
- Se muestran y describen las clases propuestas para cada capa de modelo DVO (capítulo 4).
- Se muestran y describen las estrategias de distribución definidas en la capa de control del modelo DVO (capítulo 4).
- Se muestran y describen los métodos remotos básicos definidos para el protocolo DVO, empleados por la capa de comunicación del modelo (capítulo 4).
- Se hizo la formalización matemática de las definiciones del paradigma orientado a objetos (capítulo 4).
- Se hizo la formalización matemática de los conceptos que definen a los objetos (capítulo 4).
- Se hizo la formalización matemática de las definiciones de los objetos remoto, distribuido y visual (capítulo 4).
- Se hizo la formalización matemática de la definición de los objetos distribuidos visuales, dados por la composición de los objetos distribuidos y los objetos visuales (capítulo 4).
- Se realizó la demostración del Teorema 1 (De la disminución del tamaño del mensaje usando DVO) (capítulo 4).
- Se implementó un caso de estudio empleando el modelo DVO (capítulo 5).
- Se hizo la caracterización del modelo DVO en un visor de imágenes (capítulo 5).
- Se diseño un grafo dirigido de estado para el visor de imágenes (capítulo 5).
- Se empleo la metodología OMT para la descripción del visor de imágenes (capítulo 5).
- Se definieron las clases utilizadas en el visor de imágenes empleando el modelo DVO (capítulo 5).
- Se realizaron pruebas de desempeño del visor de imágenes (capítulo 5).

## 6.2. Discusión

En esta tesis presentamos un modelo de objeto distribuido visual basado en capas. Este modelo lo creamos como base para la creación de un manejador de ventanas distribuido. Además proporcionamos una implementación de un visor de imágenes sobre la pared de vídeo usando el modelo DVO. Implementamos técnicas de distribución de datos para realizar el despliegue de las imágenes en forma distribuida en cada nodo del cluster de visualización.

Hemos encontrado diversos problemas a resolver, en cuanto al desempeño de la pared de vídeo. Muchas de ellas se mencionan en los diversos trabajos leídos. Buscamos soluciones que resuelven cada uno de estos problemas por separado. Unos basados en hardware y la mayoría busca la solución en el software; ya que es más económico. Encontramos que dentro de estos trabajos no hay soluciones que trabajen en general sino de manera específica. La mayoría trabaja sólo con objetos 3D y con la biblioteca Open GL. Es por eso que proponemos un modelo que permita el despliegue de diversas visualizaciones, de modo que sea posible la generación de aplicaciones de tipos variados mediante el uso del modelo de objetos distribuidos visuales.

### 6.2.1. Comunicación

Una de las partes de mayor interés para la implementación de este modelo de objetos distribuidos visuales, involucra la mensajería que se debe emplear. Ya sea para el envío del dato inicial ( que es el dato más grande en el caso de la replicación), como en el de los eventos que se manejen. Tomando como caso base la replicación del estado de los objetos, verificamos este tipo de envío se realiza de un modo más constante en tiempo. Para este tipo de ambientes de despliegue se espera que el tiempo en todos los nodos de despliegue (cluster de visualización) sea constante, con la finalidad de desplegar en cada nodo al mismo tiempo o en un tiempo promedio similar mínimo. Dentro de las implementaciones de los prototipos de visor de imágenes. Realizamos una comparación entre la notificación y la invocación remota. Los resultados mostraron como el tiempo de despliegue del cluster de visualización con la notificación fue mejor que con la invocación remota. A continuación trataremos de explicar por qué sucedió así.

En la (ver figura 6.1 ) se muestra como se emplea la invocación remota, en este caso cada mensaje es enviado individualmente a cada servidor. El tiempo de entrega es muy variado y en nuestro caso no permite el despliegue al mismo tiempo en todos los nodos.

La notificación (ver figura 6.2) permite una entrega constante ya que envía a todos los nodos al mismo tiempo. En este caso permite un despliegue más constante que la invocación remota.

De modo que al enviar el dato inicial mediante una notificación este llega de manera constante a todos los nodos de despliegue. En un futuro es deseable emplear Broadcast en esta etapa y mejorar el tiempo de entrega del paquete.

Cuando recibimos un evento que involucre el movimiento del objeto visual es posible identificar los nodos que requieran el cambio. De modo que sería mejor enviar la invocación

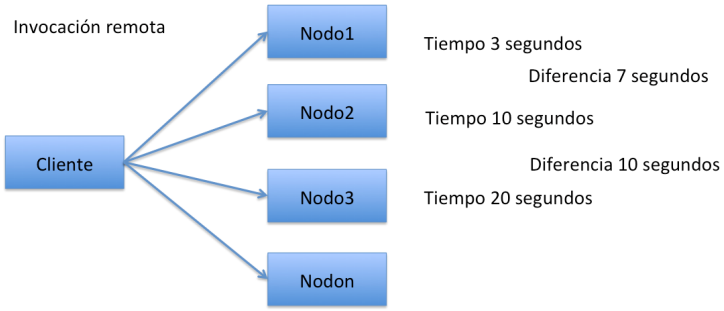


Figura 6.1: Invocación uno a uno de nodo principal(cliente) a nodos despliegue (cluster de visualización)

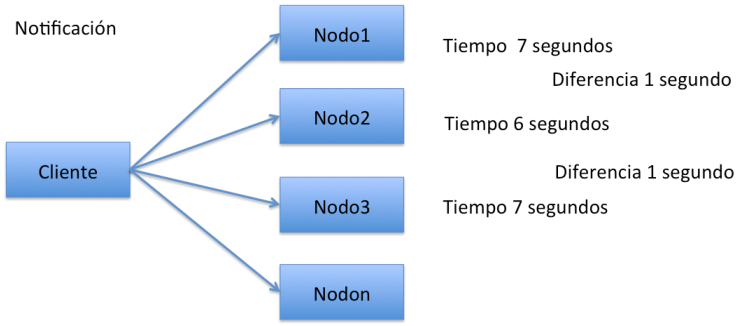


Figura 6.2: Notificación del nodo principal(cliente) a los nodos despliegue (cluster de visualización)

remota a los nodos involucrados en el cambio.

Realizamos numerosas pruebas a nivel de red con imágenes de diferentes tamaños. El resultado que tiene más relevancia en nuestro caso involucra el envío y el despliegue final de la imagen. Es por eso que en estas pruebas se toma en cuenta el tiempo total desde que se envía la imagen hasta que se despliega en todos los nodos (por lo que el número de pruebas tuvo que reducirse a 20 para 3 tipos de imágenes, por el desgaste en pantallas que un número mayor implicaría). Para ser enviada la imagen es convertida en un objeto serializado proporcionado por Objetive-C llamado NSData.

Los resultados obtenidos mostraron que cuando el envío se realiza mediante la notificación la diferencia en tiempo de recepción de cada nodo es más constante que en el caso de la invocación remota. También se generan mayores problemas de despliegue cuando la resolución incrementa y el tamaño de memoria disminuye. Por lo que se sugiere el uso de la notificación en el envío de los datos iniciales y la invocación remota cuando el tipo de mensaje sea corto. Por ejemplo, enteros que contengan el identificador de un evento y los cambios de coordenadas de un objeto.

| Resolución de imagen | Invocación remota | Notificación |
|---|---|---|
| (11477x7965) 40.7MB | Tiempo total: 1 minuto 45 segundos Tiempo de llegada por nodo: (3-44 segundos ) | Tiempo total: 1 minuto 54 segundos Tiempo de llegada por nodo: (5-9 segundos ) |
| (18001x11438) 21.4MB | Tiempo total: 34 minutos 2 segundos Tiempo de llegada por nodo: (1:03-33:01 minutos) | Tiempo total: 37 minutos 2 segundos Tiempo de llegada por nodo: (1:15-5:00 minutos) |
| (24000x12000) 6.5MB | Tiempo total: 47 minutos 15 segundos Tiempo de llegada por nodo: (1:38-2:41 minutos) | Tiempo total: 40 minutos 49 segundos Tiempo de llegada por nodo: (2:13-3:15 minutos) |

Figura 6.3: Pruebas de comunicación realizadas en el CinvesWall

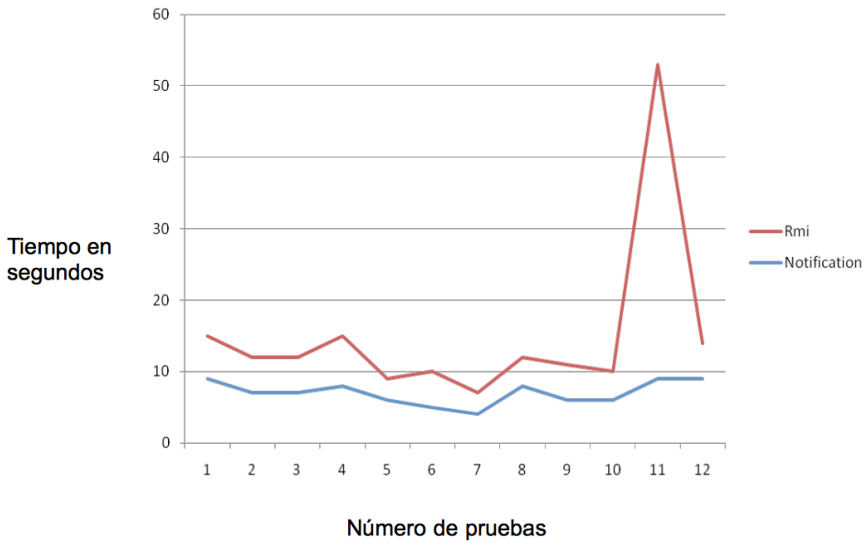


Figura 6.4: Pruebas con imagen 11477x7965

A continuación se muestran los tamaños calculados para las imágenes de prueba:

- Resolución: 11477x7965 = 91,414,305 = 91 Mp x 32 = 2,925,257,760 = 3 Gb: Tamaño

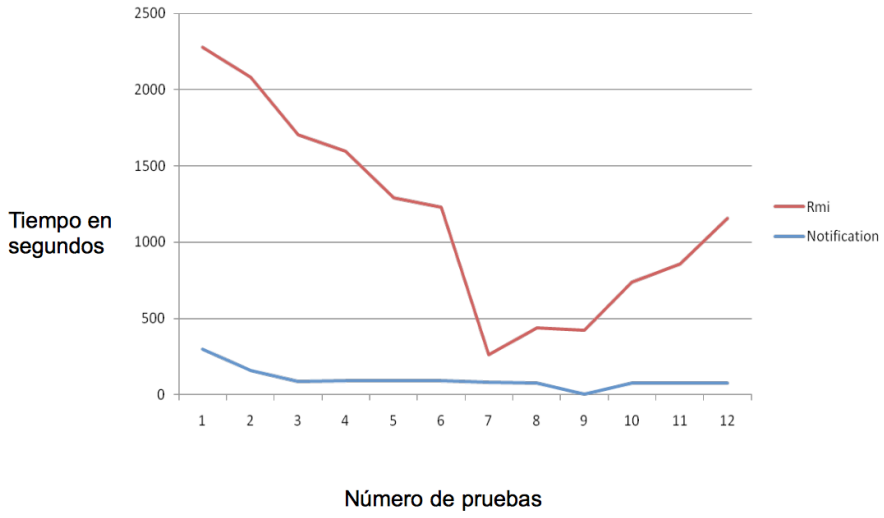


Figura 6.5: Pruebas con imagen 18001x11438

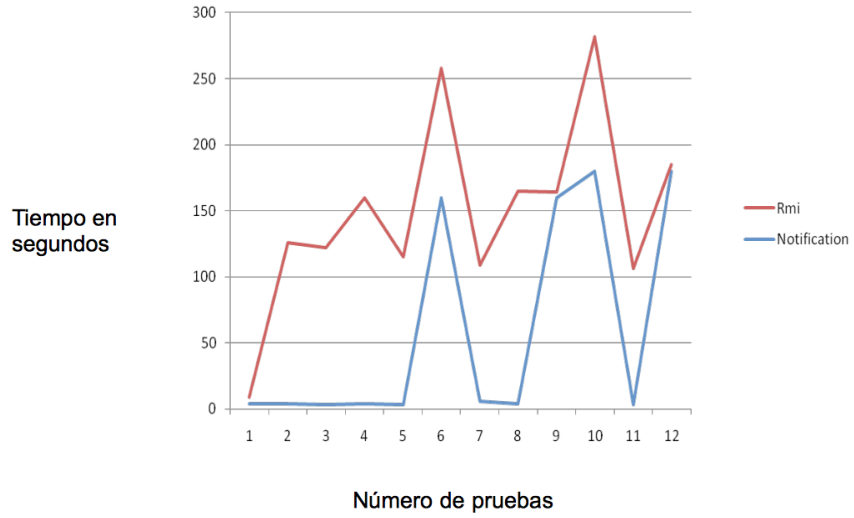


Figura 6.6: Pruebas con imagen 24000x12000

de la imagen (paquete sin compresión).

- Resolución: 18001x11438 = 205,895,438 = 205 Mp x 32 = 6,588,654,016 = 6.136 Gb: Tamaño de la imagen (paquete sin compresión).
- Resolución: 24000x12000 = 288,000,000 = 288 Mp x 32 = 9,216,000,000 = 8.583

Gb: Tamaño de la imagen (paquete sin compresión).

Hay que considerar que otra opción es enviar la imagen comprimida y que cada nodo la descomprima. Pero por la arquitectura el servidor de visualización tiene mayor capacidad que cada nodo y el tiempo que tarden en descomprimir será mayor.

### 6.2.2. Distribución

Para realizar la distribución del estado del objeto existen diversos modos. En este caso elegimos la replicación ya que nos permite obtener un manejo mas flexible de toda el área de despliegue de la pared de video. La replicación es una técnica empleada usualmente cuando se requiere mantener la funcionalidad del objeto en diferentes servidores (ver figura 6.7). En este caso se acopla con el modelo permitiendo, no tener que enviar el dato inicial en cada ocasión. Recordemos que el dato inicial es un vídeo, una imagen o un modelo 3D. Es por eso que conviene más mantener el objeto en cada servidor y desplegarlo cuando sea necesario. Permitiendo realizar cambios a través de datos pequeños como un par de coordenadas.

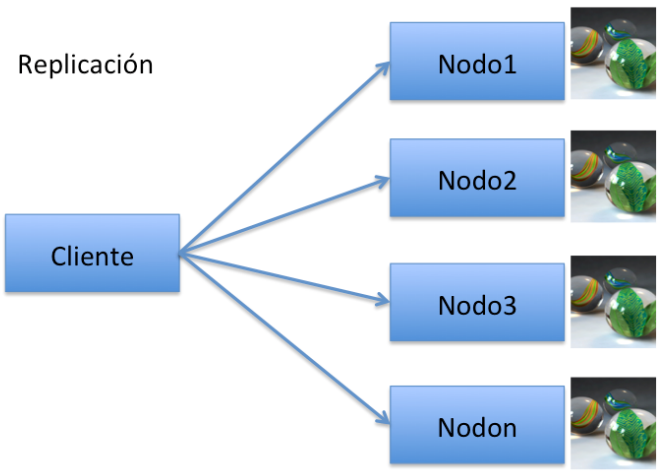


Figura 6.7: Replicación de objetos

Otra estrategia de distribución de datos es la fragmentación y es claramente una ventaja en el envío del paquete inicial. El paquete es una fracción del dato original (ver figura 6.8). El problema inicia al realizar algún movimiento entre los servidores, ya que sería necesario enviar la fracción requerida para cada servidor. Esto requiere mayor procesamiento del lado del cliente (servidor de visualización) y puede generar cuellos de botella al realizar un movimiento largo.

Emplear un híbrido de replicación-fragmentación (ver figura 6.9), permitirá tener el objeto inicial en todos los servidores y desplegar solo un fragmento de la imagen, esto va

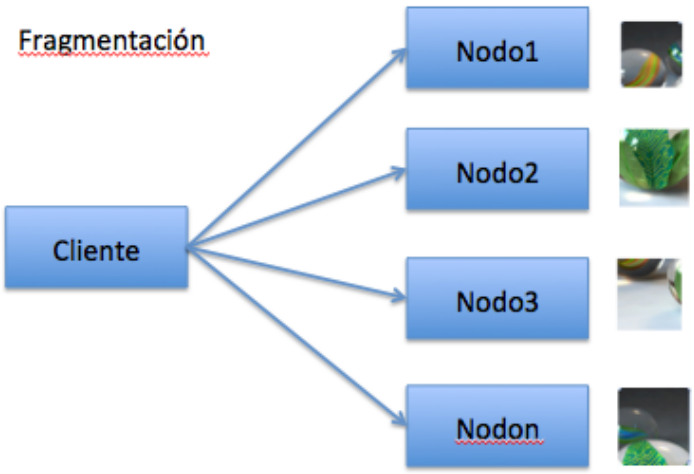


Figura 6.8: Fragmentación de objetos

a requerir mayor procesamiento en los servidores y el control mediante una estrategia de fragmentación actualización, en cada movimiento.

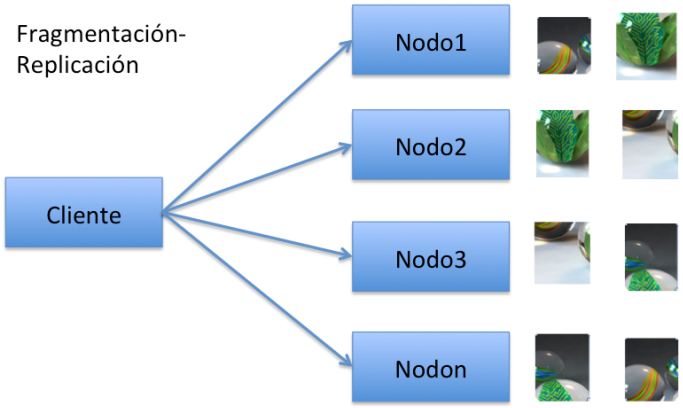


Figura 6.9: Fragmentación-replicación de objetos

### 6.2.3. Despliegue

En esta parte se involucra el modo de implementación del objeto visual que debe en todo momento estar visible y listo para recibir los eventos del usuario. Como ya se menciono en las secciones anteriores, el como va a llegar la información y el como se va a distribuir, podemos llegar al como se va a desplegar. En este caso buscamos estar lo más cerca del

GPU, de ahí que sea necesario contabilizar el tamaño de cada frame buffer de los nodos despliegue. Para lograr emplear los beneficios de la plataforma en la que trabajamos, localizamos una biblioteca que tiene acceso directo al núcleo del sistema y al GPU, llamada Quartz. Para su uso es posible manejar a las imágenes como texturas, y a los vídeos como un conjunto de imágenes, empleados dentro de un contexto de OpenGL.

## 6.3. Conclusiones

Tomando como base los resultados y las discusiones descritas anteriormente, llegamos a las siguientes conclusiones:

- Es posible clasificar a los manejadores de vídeo en dos subclases principales: render paralelo y memoria compartida.
- Es posible identificar funciones generales empleadas dentro de los manejadores de paredes de vídeo, las cuales son independientes del hardware utilizado y de las aplicaciones finales.
- Se puede definir un modelo manejador de ventanas que cumpla con las funciones generales mediante capas, donde cada capa realiza una función general.
- El modelo DVO ayuda a disminuir el tamaño del mensaje requerido para el manejo de las paredes de vídeo.
- La replicación del estado de los objetos permite el envío de los mensajes de un modo más constante en tiempo. Para este tipo de ambientes se espera que el tiempo en todos los nodos de despliegue (cluster de visualización) sea constante, con la finalidad de desplegar en cada nodo al mismo tiempo o en un tiempo promedio similar mínimo.
- El tiempo de despliegue del cluster de visualización con la notificación fue mejor que con la invocación remota.
- Cuando el envío se realiza mediante la notificación la diferencia en tiempo de recepción de cada nodo es más constante que en el caso de la invocación remota. También se generan mayores problemas de despliegue cuando la resolución incrementa y el tamaño de memoria disminuye. Por lo que se sugiere el uso de la notificación en el envío de los datos iniciales y la invocación remota cuando el tipo de mensaje sea corto.
- Los objetos distribuidos permiten la comunicación remota mediante el uso de mensajes entre objetos, es posible utilizar estos mensajes para controlar cualquier tipo de cluster, específicamente un cluster de visualización.

- El componente principal de un manejador de ventanas standard es un objeto visual, el cual permite el despliegue de cualquier tipo de información y es capaz de recibir y reconocer eventos de usuario.

- La composición de los objetos distribuidos y los objetos visuales proporciona la funcionalidad de un objeto distribuido visual.

- El objeto distribuido visual permite la construcción de un manejador de ventanas distribuido.

## 6.4. Trabajo a futuro

Durante el desarrollo de esta investigación surgieron diversos problemas y funcionalidades que no pudieron ser completados, a continuación se hace un recuento de ellos anexando las posibles soluciones:

- En un futuro es deseable emplear Broadcast en esta etapa y mejorar el tiempo de entrega del paquete. Cuando recibimos un evento que involucre el movimiento del objeto visual es posible identificar los nodos que requieran el cambio. De modo que serÌa mejor enviar la invocación remota a los nodos involucrados en el cambio.

- En un futuro se requiere verificar una arquitectura con un mejor procesamiento en cada nodo, con la finalidad de enviar los datos iniciales comprimidos. De forma que cada nodo obtenga los datos y los descomprima.

- El uso de una implementación con nodos sincronizados permitiría evitar la administración centralizada. La obtención de datos directa desde cada nodo y procesamiento sincronizado de cada aplicación puede mejorar el desempeño del manejador de la pared de vídeo.

- La amplia gama de aplicaciones que se pueden desarrollar sobre una pared de vídeo, hace que sea difícil considerar un estándar para la definición de las interfaces. Más aún si agregamos que los formatos de entrada de información a visualizar es muy diversa y que además existe una gran cantidad de bibliotecas gráficas que trabajan con sus propios formatos. De tal manera que se requiere un traductor de formatos y de instrucciones de bibliotecas gráficas. Una solución a este problema es utilizar un compilador que reconozca las instrucciones dependiendo de la biblioteca de origen. Dentro de los analizadores del compilador el análisis léxico sería el mismo para todas, el analizador sintáctico no cambiaría y sería la semántica la que nos permitiría la transformación de cualquier instrucción. En este enfoque la problemática es manejar suficientes instrucciones para cubrir las de las bibliotecas a traducir, y en caso necesario obtener combinaciones de instrucciones equiparables a una o más instrucciones.

- Los middlewares son el resultado de la búsqueda de implementaciones multiplataforma, pero a pesar de los esfuerzos realizados en el diseño y desarrollo de middlewares no se ha logrado obtener especificaciones que cumplan con todas las expectativas. Por ejemplo CORBA es uno de los principales y más robustos middlewares existentes, pero su complejidad de uso lo convierte en una herramienta poco amigable. La máquina Virtual de Java por su parte es ampliamente usada gracias a sus facilidades, pero el desempeño de Java en algunos desarrollos es considerablemente menor que en middlewares mas completos como CORBA. En particular para que los DVO puedan funcionar como un middleware hace falta mantener el desarrollo independiente de la plataforma. Una opción es mediante el uso de un IDL (Lenguaje de descripción de interfaz) definido para cada plataforma.

# Referencias


[Ahrens et al., 2001] Ahrens, J., K., B., K., M., B., G., C.C., L., and M, P. (2001). A rendering framework for multiscale views of 3d models. *Computer Graphics and Applications, IEEE , vol.21, no.4, pp.34-41, Jul/Aug.*

[Arbab et al., 1992] Arbab, F., I., H., and P., S. (1992). Interaction management of a window manager in manifold. *Computing and Information, 1992. Proceedings. ICCI '92., Fourth International Conference on.*

[Austrem, 2008] Austrem, P. G. (2008). Intelligent subject – adapting observer with push model and filters to handle divergent update needs. *EuroPLoP.*

[Baudisch et al., 2003] Baudisch, DeCarlo, A.T., D., and W.S, G. (2003). Focusing on the essential: Considering attention in display design. *Comunications of the ACM.*

[Beringer and Hullermeier, 2006] Beringer, J. and Hullermeier, E. (2006). Online clustering of parallel data streams. *Data  Knowledge Engineering pp 180-204.*

[Bhaniramka et al., 2005] Bhaniramka, P., Robert, P. C., and Eilemann, S. (2005). Opengl multipipe sdk: A toolkit for scalable parallel rendering. *IEEE Visualization 2005.*

[Blanchette and Summerfield, 2008] Blanchette, J. and Summerfield, M. (2008). C++ gui programming with qt 4 second edition. *Prentice Hall.*

[Callaghan et al., 2002] Callaghan, O., Mishra, A., M., S., G., and R, M. (2002). Streaming-data algorithms for high-quality clustering. *Proceedings of the 18th International Conference on Data Engineering, February 26-March 01.*

[Chen et al., 2001a] Chen, H., Chen, Y., Finkelstein, A., Funkhouser, T., Li, K., Liu, Z., Samanta, R., and Wallace, G. (2001a). Data distribution strategies for high-resolution displays. *Computers  Graphics, oct, vol 25, No 5, pp 811-818.*

[Chen et al., 2001b] Chen, Y., Chen, H., Clark, D. W., Liu, Z., Wallace, G., and Li, K. (2001b). Software environments for cluster-based display systems. *CCGRID '01*

*Proceedings of the 1st International Symposium on Cluster Computing and the Grid IEEE Computer Society Washington.*

[Chi, 2000] Chi, E. H. (2000). A taxonomy of visualization techniques using the data state reference model. *Xerox Palo Alto Research Center 3333 Coyote Hill Road, Palo Alto, CA 94301, pp 69-75.*

[Christian et al., 2009] Christian, P., Manuela, W., and Dieter, S. (2009). Deskotheque: Improved spatial awareness in multi-display environments. *Virtual Reality Conference, 2009. VR 2009. IEEE , vol., no., pp.123-126, 14-18 March.*

[Coulouris et al., 2007] Coulouris, G., Dollimore, J., and Kindberg, T. (2007). Distributed systems concepts and design. *Addison Wesley, Chapter 5, pages 162-163.*

[Coutinho et al., 2006] Coutinho, M., C., A., and J., R. (2006). Vitral - a text mode window manager for real-time embedded kernels. *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on , vol., no., pp.1254-1260, 20-22 Sept.*

[Czerwinski et al., 2003] Czerwinski, Robertson, B., M., and Robbins, S. G. (2003). Toward characterizing the productivity benefits of very large displays. *In Proceedings of Interact pp.9-16.*

[Czerwinski et al., 2005] Czerwinski, M., Robbins, D., Tan, D., Robertson, G., Meyers, B., and Smith, G. (2005). Large display research overview. *Microsoft Research One Microsoft Way.*

[DeFanti et al., 2009] DeFanti, T. A., Leigh, J., Renambot, L., Jeong, B., Verlo, A., and et al., L. L. (2009). The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Generation Computer Systems 25, pp 114-123.*

[Doerr and Kuester, 2011] Doerr, K.-U. and Kuester, F. (2011). Cglx: A scalable, high-performance visualization framework for networked display environments. *Visualization and Computer Graphics, IEEE Transactions on.*

[Ebara, 2011] Ebara, Y. (2011). Experiment on multi-video transmission with multipoint tiled display wall. *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing.*

[Eilemann et al., 2008] Eilemann, S., Makhinya, M., , and Pajarola, R. (2008). Equalizer: A scalable parallel rendering framework. *Proceeding SIGGRAPH Asia '08 ACM SIGGRAPH ASIA 2008 courses.*

[Faith and Martin, 2003] Faith, R. E. and Martin, K. E. (2003). Scaled window suppor t in dmx. *Copyright 2003 by Red Hat, Inc., Raleigh, North Carolina.*

[Flavell, 2010] Flavell, L. (2010). Beginning blender open source 3d modeling and animation and game design. *Apress.*

[Foley et al., 1997] Foley, J. D., Andries van Dam, S. K. F., and Hughes, J. F. (1997). Computer graphics principles and practice. *Addison Wesley.*

[Freeman et al., 2004] Freeman, E., Freeman, E., Bates, B., sierra, K., and Robson, E. (2004). Head first design patterns. *O Reilly.*

[Fu and Hanson, 2007] Fu, C. and Hanson, A. (2007). A transparently scalable visualization architecture for exploring the universe. *IEEE Transactions on Visualization and Computer Graphics.*

[Funkhouser and Li, 2000] Funkhouser, T. and Li, K. (2000). Large-format displays. *Computer Graphics and Applications, IEEE, pp 20 - 21.*

[Gamma et al., 1997] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1997). Design patterns elements of reusable object-oriented software. *Addison-Wesley.*

[Gettys et al., 1990] Gettys, J., Karlton, P. L., and McGregor, S. (1990). Scaled window suppor t in dmx. *Digital Equipment Corporation.*

[Goglin, 2008] Goglin, B. (2008). Design and implementation of open-mx: High-performance message passing over generic ethernet hardware. *Workshop on Communication Architecture for Clusters.*

[Goglin, 2009] Goglin, B. (2009). Decoupling memory pinning from the application with overlapped on-demand pinning and mmu notifiers. *Workshop on Communication Architecture for Clusters.*

[Grosso, 2002] Grosso, W. (2002). Java rmi. *O'Reilly.*

[Hericko and Beloglavec, 2005] Hericko, M. and Beloglavec, S. (2005). A composite design-pattern identification technique. *Informatica 29 (2005) 469–476 469.*

[Hodges, 2003] Hodges, R. S. (2003). The guild handbook of scientific illustration. *John Wiley and Sons. ISBN 0-471-36011-2.*

[Hsu et al., 2011] Hsu, W., Ma, K., and Correa, C. (2011). A rendering framework for multiscale views of 3d models. *ACM Transactions on Graphics 30(6) pp 131:1-10.*

[Huang et al., 2006] Huang, E., Mynatt, E., Russell, D. M., and Sue, A. E. (2006). Secrets to success and fatal flaws: The design of large display groupware. *IEEE Computer Graphics, Applications, pp 37-45.*

[Huang, 2005] Huang, E. M. (2005). Mechanisms for collaboration: A design and evaluation framework for multi-user interfaces. *In CHI '05 extended abstracts on Human factors in computing systems (2005), pp. 1118-1119.*

[Humphreys et al., 2001] Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., and Hanrahan, P. (2001). Wiregl: a scalable graphics system for clusters. *SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques.*

[Humphreys et al., 2000] Humphreys, G., Eldridge, M., Buck, I., Stoll, G., and Hanrahan, P. (2000). Distributed rendering for scalable displays. *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), p.30-es, November 04-10.*

[Humphreys et al., 2002] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. (2002). Chromium: a stream-processing framework for interactive rendering on clusters. *2002 Proceedings of the 29th annual conference on Computer graphics and interactive techniques.*

[Inc., 2009] Inc., A. (2009). Nsview class reference user experience: Windows views. *Apple Inc. All Rights Reserved.*

[Jeong et al., 2010] Jeong, B., Leigh, J., Johnson, A., L., R., M., B., R., J., S., N., and H., H. (2010). Ultrascale collaborative visualization using a display-rich global cyberinfraestructure. *IEEE Computer Graphics and Applications, pp 71-83.*

[Kang and Chae, 2007] Kang, Y.-B. and Chae, K.-J. (2007). Xmegawall: A super high-resolution tiled display using a pc cluster. *Proceedings of Computer Graphics International, Petropolis, Brazil, 30 May to 2 June, pp. 29-36.*

[Krasner and Pope, 1988] Krasner, G. E. and Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *ParcPlace Systems, Inc.*

[Krishnaprasad et al., 2004] Krishnaprasad, N. K., Vishwanath, V., Venkataraman, S., Rao, A. G., Renambot, L., Leigh, J., and Johnson, A. E. (2004). Juxtaview - a tool for interactive visualization of large imagery on scalable tiled displays. *Proceeding CLUSTER '04 Proceedings of the 2004 IEEE International Conference on Cluster Computing.*

[Kukimoto Nobuyuki, 2010] Kukimoto Nobuyuki, Guillaume Fleury, J. L. (2010). Tiled display system for improved communication efficiency. *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing.*

[Kurtenbach and Fitzmaurice, 2005] Kurtenbach, G. and Fitzmaurice, G. (2005). Applications of large displays. *IEEE CGA.*

[Lauesen, 2005] Lauesen, S. (2005). User interface design, a software engineering perspective. *Addison Wesley.*

[Lawlor et al., 2008] Lawlor, O. S., Page, M., and Genetti., J. (2008). Mpiglut: Powerwall programming made easier. *Journal of WSCG.*

[Leigh et al., 2008] Leigh, Renambot, A., J., R., J., Hur, H., E., H., and D., L. (2008). Scalable adaptive graphics middleware for visualization streaming and collaboration in ultra resolution display environments. *This paper appears in: Ultrascale Visualization, 2008. UltraVis 2008. Workshop on.*

[Leigh et al., 2006] Leigh, J., Renambot, L., Johnson, A., Jeong, B., and Ratko Jagodic, e. a. (2006). The global lambda visualization facility: An international ultra-high-definition wide area. *Visualization Collaboratory Future Generation Computer Systems.*

[Lipsa et al., 2011] Lipsa, D., Laramee, R., Bergeron, R., and Sparr, T. (2011). Techniques for large data visualization. *J. of Research and Reviews in Computer Science 2(2):315-322.*

[McCormack and McNamara, 1993] McCormack, J. and McNamara, B. (1993). A smart frame buffer. *WRL Research Report 93/1.*

[MCGuffin, 2002] MCGuffin, M. J. (2002). Fitts' law and expanding targets: An experimental study, and applications to user interface design. *Graduate Department of Computer Science University of Toronto.*

[Michnick and Tarassov, 2005] Michnick, S. W. and Tarassov, K. (2005). ivici: Interrelational visualization and correlation interface. *Published online 2005 December 30. doi: 10.1186/gb-2005-6-13-r115.*

[Molnar et al., 1994] Molnar, S., Cox, M., Ellsworth, D., and Fuchs, H. (1994). A sorting classification of parallel rendering. *IEEE CGA.*

[Mueller, 1995] Mueller, C. (1995). The sort-first rendering architecture for high-performance graphics. *Proceedings of the 1995 symposium on Interactive 3D graphics.*

[Myers, 1995] Myers, B. (1995). User interface software tools. *Transactions on Computer-Human Interaction (TOCHI).*

[Myers et al., 2000] Myers, B., Hudson, S., and Pausch, R. (2000). Past, present, and future of user interface software tools. *Computer-Human Interaction (TOCHI).*

[Myers, 1998] Myers, B. A. (1998). A brief history of human computer interaction technology. *ACM interactions. Vol. 5 no. 2 March pp. 44-54.*

[Naiksatam et al., 2005] Naiksatam, S., Figueira, S., Chiappari, S. A., and Bhatnagar, N. (2005). Analzing the advantage reservation of lightpaths in lambda-grids. *CCGrid.*

[Nguyen et al., 2011] Nguyen, H., Abramson, D., Bethwaite, B., Dinh, M. N., Enticott, C., Garic, S., Russel, A., Firth, S., Harper, I., Lackmann, M., and Vail, M. (2011). Integrating scientific workflows and large tiled display walls: Bridging the visualization divide. *International Conference on Parallel Processing Workshops.*

[Ni et al., 2006] Ni, T., Schmidt, G. S., Staadt, O. G., Livingston, M. A., Ball, R., and May, R. (2006). A survey of large high-resolution display technologies, techniques, and applications. *Virtual Reality Conference, 2006.*

[Norman, 1998] Norman, D. A. (1998). The design of everyday things. *The MIT Press.*

[Nouanesengsy et al., 2011] Nouanesengsy, Lee, T.-Y., and Shen, H.-W. (2011). A rendering framework for multiscale views of 3d models. *Visualization and Computer Graphics, IEEE Transactions on , vol.17, no.12, pp.1785-1794, Dec.*

[Nutting et al., 2009] Nutting, J., Mark, D., and LaMarche, J. (2009). Learn cocoa on the mac. *Apress.*

[Object Management Group, 2011] Object Management Group, I. (2011). Common object request broker architecture (corba). *Specification, Version 3.2 Part 1: CORBA Interfaces Copyright ®.*

[Papadopoulos et al., 2004] Papadopoulos, P. M., Papadopoulos, C. A., Katz, M. J., Link, W. J., , and Bruno, G. (2004). Configuring large high-performance clusters at lightspeed: A case study. *International Journal of High Performance Computing Applications archive Volume 18 Issue 3, August.*

[Puder et al., 2006] Puder, A., Romer, K., and Pilhofer, F. (2006). Distributed systems architecture a middleware approach. *Morgan Kaufmann Publishers, Elsevier, Chapter 2, pages 16-18.*

[Ramirez et al., 2012] Ramirez, R. L., S.V.C, V., and M, V. A. (2012). Visual data mining over a video wall. *Electrical Communications and Computers (CONIELECOMP), 2012 22nd International Conference on Date of Conference: 27-29 Feb. 2012.*

[Renambot et al., 2008] Renambot, L., Jeong, B., Hur, H., johnson, A., and Leigh, J. (2008). Enabling high resolution collaborative visualization in display rich virtual. *Electronic Visualization Laboratory, University Illinois at Chicago.*

[Richardson et al., 1998] Richardson, Stafford-Fraser, K.R., W., and A, H. (1998). Virtual network computing. *Internet Computing, IEEE , vol.2, no.1, pp.33-38, Jan/Feb.*

[Richardson, 2010] Richardson, T. (2010). Rfb protocol. *Olivetti Research Ltd / ATT Labs Cambridge.*

[Robertson et al., 2000] Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., Thiel, D., and Gorokhovsky, V. (2000). The task gallery: A 3d window manager. *CHI 2000 I-6 APRIL 2000.*

[Sakuraba et al., 2011] Sakuraba, A., Ishida, T., and Shibata, Y. (2011). A new interface for large scale tiled display system considering scalability. *2011 International Conference on Network-Based Information Systems.*

[Saund and Moran, 1995] Saund, E. and Moran, T. P. (1995). Perceptual organization in an interactive sketch editing application. *Xerox Palo Alto Research Center*.

[Shneiderman, 2000] Shneiderman, B. (2000). Creating creativity: user interfaces for supporting innovation. *Transactions on Computer-Human Interaction (TOCHI)*.

[Singh et al., 2003] Singh, R., Leigh, J., DeFanti, T. A., and Karayannis, F. (2003). Teravision: a high resolution graphics streaming device for amplified collaboration environments. *Published in: Journal Future Generation Computer Systems - iGrid 2002 archive*.

[Smarr et al., 2003] Smarr, L. L., Chien, A. A., DeFanti, T., Leigh, J., and Papadopoulos, P. M. (2003). The optiputer. *Communications of the ACM Volume 46 Issue 11, November 2003*.

[Tanenbaum and Steen, 2008] Tanenbaum, A. S. and Steen, M. V. (2008). Distributed systems principles and paradigms. *Prentice Hall, Chapter 10, pages 444-445, 2008*.

[van Steen M. et al., 2002] van Steen M., P., H., and A.S, T. (2002). Globe: a wide area distributed system. *Dept. of Math. and Comput. Sci., Vrije Univ., Amsterdam, Netherlands*.

[Verta et al., 2005] Verta, O., Talia, D., and PaoloTrunfio (2005). Weka4ws: a wrsf-enabled weka toolkit for distribuited data mining on grids. *Data Mining Grid digital library*.

[Vishwanath, 2009] Vishwanath, V. (2009). Lambdaram a high-performance, multi-dimensional, distributed cache over ultra-high speed networks. *PHD thesis B.E. University of Illinois at Chicago*.

[Wang and Qian, 2010] Wang, R. and Qian, X. (2010). Openscenegraph 3 beginner ' s guide. *Packt Publishing Ltd*.

[Wilson, 1998] Wilson, B. G. (1998). Constructivist learning environments: Case studies in instructional design. *In IEEE/ACM International Symposium on Cluster Computing and the Grid*.

[Wittenbrink et al., 2011] Wittenbrink, C., E., K., and A., P. (2011). Fermi gf100 gpu architecture. *Micro, IEEE , vol.31, no.2, pp.50-59, March-April*.

[Wong et al., 2007] Wong, C.-O., Kyoung, D., and Jung, K. (2007). Adaptive context aware attentive interaction in large tiled display. *Springer-Verlag Berlin Heidelberg*.

[Wright, 2007] Wright, H. (2007). Introduction to scientific visualization. *Springer*.

[Wylie et al., 2001] Wylie, Pavlakos, V., L., and K., M. (2001). Scalable rendering on pc clusters. *Computer Graphics and Applications, IEEE , vol.21, no.4, pp.62-69, Jul/Aug*.

[Yuill and Rogers, 2012] Yuill, N. and Rogers, Y. (2012). Mechanisms for collaboration: A design and evaluation framework for multi-user interfaces. *ACM Transactions on Computer-Human Interaction*.

[Zhang et al., 2003] Zhang, C. C., Leigh, J., Defanti, T. A., Mazzucco, M., and Grossman, R. (2003). Terascope: Distributed visual data mining of terascale data sets over photonic networks. *Journal of Future Generation Computer Systems (FGCS), Elsevier Science Press 19:935 – 943*.

[Zimmermann, 1980] Zimmermann, H. (1980). Os1 reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*.