# "Mejoras de ODUst: Un sistema para compartir aplicaciones"

Oscar R. Reyes,
osrehe@elo.utfsm.cl

Agustín J. González,
agustin.gonzalez@elo.utfsm.cl

Departamento de Electrónica, Universidad Técnica Federico Santa María
Casilla 110-V, Valparaíso, Chile
Teléfono (56-32) 654196, FAX (56-32) 797469

## Resumen

Este documento discute las mejoras implementadas a una herramienta para colaboración de datos, que permite a los usuarios compartir las aplicaciones que se ven en escritorio. La aplicación llamada ODUst fue desarrollada en Java para lograr independencia de la plataforma, utiliza UDP multicast para la escalabilidad y permite que usuarios remotos tomen el control de la aplicación que esta siendo compartida. Este escrito describe las mejoras respecto a la versión original de la herramienta: se eliminó la dependencia de Java Advanced Imaging que dificultaba la configuración del sistema, también se incluye un mecanismo para transmisión de forma y posición del mouse. Las mejoras introducidas hacen el sistema mucho más eficiente y permiten un trabajo más productivo a los usuarios.

## Palabras claves

Herramienta de software, aplicaciones compartidas, mejoras.

## 1 Introducción

La necesidad de comunicación ha crecido conjuntamente con el ancho de banda de las redes. Así también ha aparecido toda una generación de herramientas de comunicación online. En un comienzo y aún en la actualidad el mayor auge se produce en las llamadas herramientas asincrónicas o aquellas donde los usuarios no interactúan al mismo tiempo, entre ellas se destacan el correo electrónico, pasando por foros, páginas Web, presentaciones multimedia, cursos interactivos y últimamente los weblog [1]. A pesar que estas cumplen una tarea específica y resultan muy útiles, no pueden suplir por ejemplo la necesidad de comunicación en tiempo real entre grupos usuarios, como conferencias, colaboración y trabajo compartido. Para este tipo de prácticas existen afortunadamente aunque menos común, un grupo reducido de herramientas de colaboración en vivo llamadas sincrónicas. Estas han sido muy utilizadas para el aprendizaje electrónico, particularmente para conferencias y capacitación a distancia. Las aplicaciones destinadas a estos fines requirieren transmisión de audio, vídeo y datos, mientras la transmisión de video y audio da origen a video-conferencias y audio-conferencias la transmisión de datos da origen a otro tipo de aplicaciones como mensajería Instantánea, pizarras compartidas. ODUst [2], el sistema mejorado que trata este artículo es un ejemplo de un más demandado tipo de aplicaciones que permiten compartir cualquier aplicación visible en el escritorio. Ejemplos de aplicaciones distribuidas sincrónicas VIC [3] para video conferencias, RAT [4] diseñada para audio conferencias, y Messenger [4], cuya principal característica es la mensajería instantánea, también proporciona video y audio aunque punto a punto. Dentro de la gama de herramientas que comparten aplicaciones podemos destacar a VNC [5] cuya función es la de ejecutar el escritorio remotamente y permitir a los usuarios ejecutarlo como si estuvieran localmente. NetMeeting [6] actualmente integrado a Messenger, también comparte aplicaciones. Los últimos dos sistemas poseen una arquitectura centralizada y usan TCP como protocolo de transporte por ese motivo están limitados a un número reducido de de usuarios típicamente menor que diez. Los sistemas de audio y video generalmente escalan muy bien utilizando UDP multicast. Por esta razón han sido fuertemente utilizados, por ejemplo en sistemas de educación a grandes grupos de participantes del orden de varias decenas. Aquí se ha utilizado sistemas de difusión de video como los ofrecidos por la empresa RealNetworks [7], un caso interesante de mencionar es el de la Universidad de California Berkeley, donde las clases se encuentran disponibles bajo demanda [http://webcast.berkeley.edu]. Una de las principales ventajas de este sistema es la escalabilidad, mientras que un inconveniente es que generalmente sistemas de esta índole son unidireccionales. En general los sistemas como el anterior no permiten la interacción directa entre grandes números de participantes en tiempo real.

ODUst es una herramienta para compartir aplicaciones sincrónicamente. Trabaja capturando continuamente la ventana de la aplicación compartida y enviándola vía UDP multicast a todos los receptores, uno de ellos puede tomar el control de la aplicación como si estuviera en su propio escritorio; naturalmente el retardo de la red no puede ser evitado. Aunque la mayor parte de la aplicación esa desarrollada en Java, algunas funciones como la captura de ventanas y la inyección de eventos de mouse y teclado son provistas por bibliotecas nativas. Esto crea dependencias que fuerzan a disponer de múltiples versiones del software. En la actualidad cualquier maquina equipada con Java puede servir de receptor, pero solo los usuarios de Windows y Linux pueden compartir sus aplicaciones. Las ventajas principales de Odust incluyen el poder llegar a un gran número de usuarios remotos y adaptarse al cambio en el número de receptores que ir y venir como deseen. Por otro lado Odust requiere soporte multicast que no esta ampliamente disponible en Internet, donde se encuentra el gran número de usurarios. Otra desventaja es que la posición del cursor y su forma no son transmitidas con las ventanas de las aplicaciones compartidas, ya que el mecanismo de captura normal de pantalla no incluye el mouse, que es usualmente dibujado sobre la pantalla por el controlador de video. Finalmente la versión inicial de Odust usa JAI (Java Advanced Imaging) [8] para comprimir y descomprimir los cuadros de video. La dependencia de este paquete causa problemas a los usuarios ya que a la gran mayoría le resulta compleja la instalación o la configuración de las variables de entorno necesarias para una correcta ejecución. Este artículo se centra en dos mejoras a Odust: la dependencia de JAI es eliminada y como funciona el nuevo mecanismo de transmisión de la figura y posición del cursor. Otras mejoras como el soporte TCP y esquemas de menor consumo de ancho de banda para la transmisión de imágenes están en desarrollo.

## 2 Arquitectura básica de ODUST

ODUst es un sistema capaz de compartir aplicaciones remotas, de modo que los participantes en una sesión de este tipo pueden colaborar simultáneamente tal como si estuvieran en un mismo espacio físico. Los participantes se reúnen virtualmente conectados a través de una red y puedan trabajar en conjunto sobre un mismo programa. La aplicación transmite la interfaz gráfica de cualquier programa que visible en pantalla a uno o más receptores conectados. Cualquiera de los integrantes puede tomar en algún momento determinado el mando de la aplicación remota para hacer su aporte. ODUst se presenta como un complemento a otro tipo software existente en el mercado y que ofrecen variantes de trabajo sobre Internet, como por ejemplo: programas de video conferencia, programas de mensajería, aplicaciones de tipo pizarra para intercambio de información gráfica como dibujos y diagramas. Las aplicaciones complementarias que resultan más útiles al momento a trabajar con ODUst son las de audio conferencia. El sistema ODUst se compone básicamente de dos partes un transmisor y un receptor, figura 1. El transmisor es el encargado de difundir una aplicación compartida, este muestrea y captura regularmente las aplicaciones en Windows, comprime la secuencia de cuadros y los envía. También recibe e inserta eventos de entrada e teclado y mouse, que son inyectados por un operador remoto. Por otro lado receptor desarrollado completamente en Java es capaz de recibir y descomprimir y desplegar, la secuencia de imágenes de las aplicaciones difundidas por diferentes transmisores. Adicionalmente transmisor y receptor implementan un mecanismo distribuido de control de piso [9], para otorgar el control sobre la aplicación de una manera exclusiva.

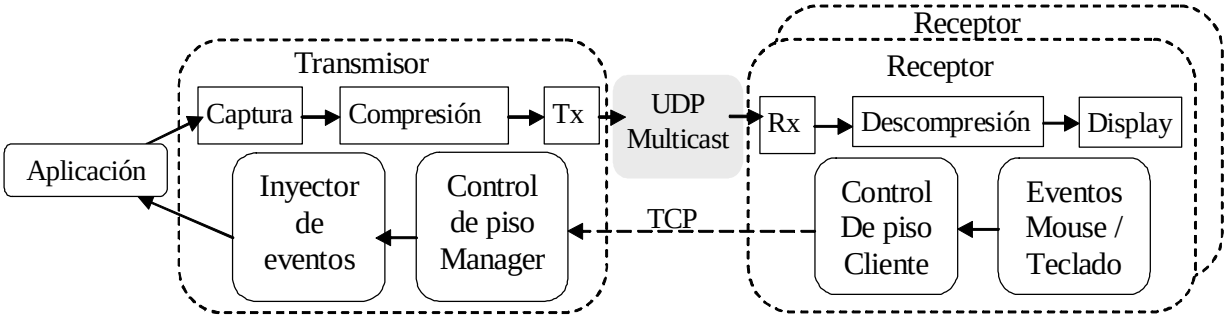


figura n°1. Arquitectura básica de Odust.

El esquema de la figura 2 muestra una sesión típica ODUst donde a través de una red multicast. Usuarios con diferentes sistemas operativos comparten sus aplicaciones. Se puede ver que el Agustín comparte una ventana de MS Word, del mismo modo Oscar comparte una ventana de Notepad, además de transmitir estos usuarios pueden ver las

aplicaciones compartidas que circulan por la red. Es decir Agustín recibe el Notepad compartido por Oscar, y éste MS Word. Por otro lado los otros dos usuarios reciben ambas aplicaciones compartidas.

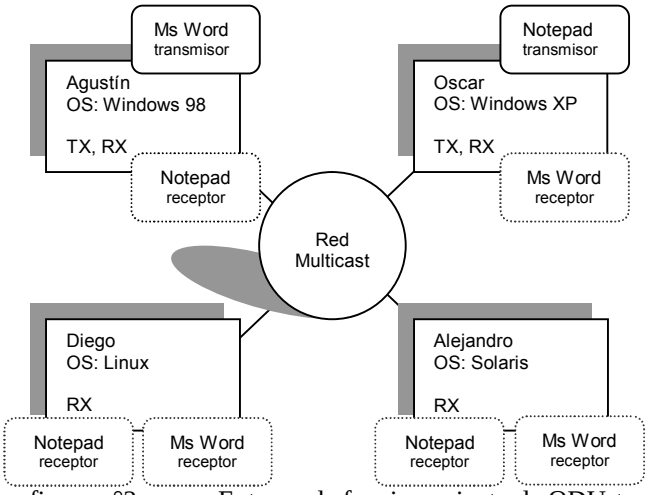


figura n°2. Entorno de funcionamiento de ODUst

La implementación del sistema está pensada fundamentalmente en la tecnología Java 2 Standard Edition (J2SE) [10]. La actual versión tiene bibliotecas nativas (para captura de ventana y inyección de eventos) para sistemas Windows y Linux.

Las tres principales características de la plataforma distribuida ODUst son la difusión de la imagen de la aplicación, control de piso para que cualquiera de los participantes tome control de la aplicación y la interacción remota.
Mientras el dueño de la aplicación compartida opera directamente los participantes ven y operan las imágenes que son enviadas por ODUst, estas resultan indistinguibles de la aplicación real. La funcionalidad esta desarrollada con "process granularity" que significa que todas las ventanas de una misma aplicación compartidas serán transmitidas automáticamente.
El mecanismo de control de piso permite a cualquier receptor pedir el control de la aplicación compartida. Aunque un receptor puede tener un piso a la vez, el propietario de la herramienta compartida puede operarla al mismo tiempo. Una desventaja de esta técnica es la interferencia de los eventos de entrada, esto es teclado y mouse, con los mismos artefactos de entrada en la aplicación del dueño de la aplicación. Debido a la naturaleza del protocolo ligero de middleware, cualquier participante puede dejar la sesión de colaboración en cualquier momento, asimismo cualquier persona puede unirse a la sesión en cualquier momento. Estas dos situaciones virtualmente no tienen efecto en los otros participantes. Los usuarios que se unen a sesión en forma tardía reciben igualmente una vista sincrónica dentro de un tiempo límite, definido por un parámetro en ODUst. Los multiples participantes pueden compartir sus aplicaciones en cualquier momento con un máximo de una por usuario, sin embargo pueden recibir todas las que circulen por la red. ODUst multiplexa el canal multicast en 256 sub-canales que permiten disponer de múltiples aplicaciones compartidas, es decir teóricamente se podría compartir ese número, pero en la práctica los recursos de las maquinas limitan éste valor a uno mucho inferior. Cada aplicación compartida es visualizada en una ventana diferente. La escalabilidad se logra utilizando transmisión del tipo multicast, aunque la aplicación también puede funcionar punto a punto entre dos participantes con otro esquema de red.
Una característica muy importante de ODUst es el protocolo dinámico de transmisión de imágenes, dado que la aplicación necesita hacer una utilización eficiente de los recursos disponibles se han implementado mecanismos para llevarlo a cabo, principalmente para manejar la redundancia. En el trasmisor la eliminación de la renuncia temporal se logra muestreando la imagen a periodos regulares (~2 Hz dependiendo de su tamaño), se divide la imagen en "Tiles" o pequeños rectángulos, se envían solo las tiles que han cambiado, después de un tiempo aleatorio se retransmite cada tile para superar pérdidas debido a la transmisión UDP. La redundancia espacial se elimina comprimiendo mediante JPEG [11] y enviando los tiles que cambiaron. El receptor recibe los datos, descomprime los tiles, actualiza la imagen desplegada.

# 3 Diseño e implementación de la modificación del sistema para incorporar mejora.

Este apartado describe las modificaciones realizadas en la versión original de ODUst, en la actualidad se trabaja principalmente en la mejora de tres características, Eliminación de dependencias de módulos de software, difusión de la imagen y posición de cursor empleados en la aplicación compartida, y algoritmo de discriminación de cambios de imagen.

## 3.1 Eliminación de dependencias de JAI

La versión original de ODUst utiliza las bibliotecas Java Advanced Imaging package (JAI) Estas componentes de software entregan la principal funcionalidad de compresión de imágenes a formato JPEG, también son utiliza para detectar los cambios gráficos producidos en las imágenes de las aplicaciones a compartidas. Formalmente las API JAI proveen un conjunto de interfaces orientadas a objeto que mantienen modelos de programación simples y de alto nivel que permiten manipular imágenes fácilmente.
A pesar que la utilización de JAI favorece levemente el rendimiento, aparecen otros problemas que se pueden categorizar como no técnicos. Dado que uno de los objetivos del desarrollo de ODUst es llegar a ser un referente, se vuelve muy importante que usuarios comunes utilicen la aplicación, sin embargo uno de los mayores problemas detectados fue la dificultad para configurar manualmente el path de las clases, y lograr una compilación exitosa. Por otro lado se tenía que la dependencia de las bibliotecas JAI provee funcionalidades mínimas que podían ser suplidas alternativamente.

Eliminar la dependencia de JAI se logra utilizando bibliotecas estándares que entregan funcionalidad similar, sin necesidad de módulos adicionales.
El tratamiento y lógica de transmisión de imágenes siguen siendo los mismos, sólo se cambia la forma, como en un principio la redundancia temporal se elimina muestreando, dividiendo la imagen en tiles y enviando los cambios, la redundancia espacial se elimina comprimiendo las imágenes. A nivel interno ODUst trabaja solo con imágenes del tipo o clase BufferedImage. Una BufferedImage es la representación de una imagen con un accesible buffer de datos. Se compone de dos partes una es el ColorModel, que provee la interpretación de color de la imagen colores y componente alpha, la otra parte es Raster que representa un arreglo rectangular de píxeles con los datos de la imagen. Las Tiles en formato BufferedImage son pasadas al nuevo codificador que realiza la tarea de compresión y luego las transmite la información.
Por otro lado el receptor recibe la imagen comprimida y la decodifica entregando como resultado tiles en formato BufferedImage, que son agrupados en un buffer mayor del tamaño de la ventana de la aplicación original, posteriormente y medida que llegan las imágenes son desplegadas.

En lugar de JAI se utiliza el paquete com.sun.image.codec.jpeg que implementa codificación y decodificación JPEG. El paquete no es parte de del núcleo de las Java APIs; sin embargo es parte de las distribuciones Sun JDK y JRE. Finalmente la con implementación JPEGImageEncoder codifica los tiles BufferedImage en flujos de datos JPEG. Los flujos de datos son escritos en flujos de salida provistos por el codificador y soportados por los mecanismos de manejo de sockets de ODUst. Muy importante resulta la utilización de parámetros para ajustar valores como la calidad de compresión.

La interfaz JPEGImageDecoder recibe un flujo de datos de entrada codificados, los decodifica en formato BufferedImage, permitiendo la manipulación inmediata de las imágenes sin hacer por ejemplo alguna transformación de formato.

## 3.2 Transmisión del cursor

Determinar estado del cursor en una aplicación en curso, entrega información adicional de esta, es posible conocer el estado en el que se encuentra. Obtener la información del cursor resulta más útil aun cuando se ejecuta o comparte una aplicación remota, los usuarios pueden centrar su atención en la información entregada por el cursor y como este cambia para cada acción, es posible interpretar las acciones correctamente. ODUst carece de un mecanismo que le permita transmitir el cursor a todos los participantes. A pesar que esta parece una sencilla tarea tiene diferentes matices que aumentan su complejidad. Java por si mismo no ofrece la posibilidad de capturar directamente la imagen del cursor actual, por otro lado su posición es capturada si éste se encuentra sobre la aplicación Java en curso. Afortunadamente las API provistas por Java Native Interface (JNI) [12] permiten la interacción con bibliotecas

nativas. Específicamente el presente trabajo captura la información del cursor para las plataformas Windows, el sistema ha sido probado en Windows 98/XP. La implementación fue desarrollada en C++ Win32 [13].

El primer paso para agregar la funcionalidad a la aplicación es recuperar la imagen de cursor, esta resulta ser simplemente un bitmap. Ésta se logró implementando una biblioteca nativa. La función GetCursorInfo definida en las bibliotecas MSDN [14] recupera la información del cursor actual, retornando un valor distinto de cero si el resultado es exitoso, la función apunta la estructura CURSORINFO que recibe la información de cursor, como por ejemplo la visibilidad, destacando un manipulador "Handle" al cursor. Dado que Windows trata a los iconos y cursor es de muy similar manera, se utiliza la función GetIconInfo para recuperar información específica de iconos y cursores, esta función es quien en definitiva pasa a otra estructura llamada ICONINFO los datos más importantes de iconos y cursores, presenta entre sus miembros dos imágenes referente al cursor, ellas son la imagen propiamente tal y una máscara que especifica una imagen denominada bitmask necesaria para dejar transparente algunas áreas del cursor.

En el siguiente trozo de código puede apreciarse la funcionalidad descrita en el párrafo anterior:

```
HBITMAP     hBitmapImage;
ICONINFO    info;
CURSORINFO  curinfo;

        curinfo.cbSize = sizeof(curinfo);

        if (GetCursorInfo(&curinfo)){
                if (GetIconInfo(curinfo.hCursor, &info) == 0)
                        return 0;
                hBitmapImage = info.hbmColor;  //info.hbnMask; ….
```

figura n°3. Extracto del código para capturar el bitmap del mouse en Windows

Si el cursor es en blanco y negro el la imagen de la máscara tiene el siguiente formato la mitad superior corresponde al AND bitmask del cursor y la mitad inferior corresponde a XOR bitmask del cursor, si el cursor es en color la mascara solo define el AND bitmask del cursor

Luego de tener disponibles las imágenes (HBITMAP*)*, se prepara el formato, información acerca dimensiones y formato de color (BITMAPINFOHEADER) para almacenar la imagen, o más específicamente traspasar la imagen del cursor capturadas a la aplicación Java.
ODUst mediante JNI pregunta al código nativo el tamaño del buffer necesario para almacenar las imágenes de cursor, éste entrega la información, con lo que ODUst en un siguiente paso crea un buffer compartido en el que finalmente el código nativo deposita la información de la imagen. La información depositada en el buffer es leída y la imagen es transformada inmediatamente por ODUst al formato BufferedImage, en las siguientes etapas todo el procesamiento es realizado en Java. Las imágenes son procesadas y el resultado es un cursor idéntico al original.

El cursor es muestreado regularmente para comprobar si ha cambiado su posición o su imagen, cada imagen es almacenada en un ArrayList en Java. Si existe una imagen nueva comparada con las disponibles en la lista, ésta se almacena. Cada imagen de cursor capturada debe ser enviada a los receptores para que estos puedan apreciarlo, por cada nueva captura se envía la información. El tamaño promedio de una imagen de cursor es de 32x32 pixeles, dada la naturaleza de las imágenes (imágenes sintéticas) son comprimidas al formato Portable Network Graphics (PNG) [15] mediante las clases provistas por Java Image I/O API [16] correspondientes al paquete javax.imageio. Particularmente la clase ImageIO actúa como codificador y decodificador. Las coordenadas del cursor son recuperadas junto con sus imágenes en un arreglo determinado para ello, esta información es enviada periódicamente por el transmisor, en los mismos paquetes que contienen de los tiles de la imagen de la aplicación compartida. Anexado también a estos paquetes viaja el índice del cursor actual en la lista de imágenes de cursor.
Mientras no se reciba un cursor oficial, el receptor despliega un cursor genérico. El receptor también maneja una lista con las imágenes de cursor de modo que estas son las imágenes que despliega dependiendo del índice que recibe.

Dado que un participante a una sesión ODUst puede engancharse en cualquier momento. Se transmiten cada cierto instante todas las imágenes de los cursores disponibles con sus respectivos índices, para ser identificados y desplegados correctamente.

Una vez que la aplicación compartida es cerrada o finalizada, el transmisor envía una señal que indica que la lista debe ser borrada, ya que empezará una nueva captura, se opta por este método ya que la forma de los cursores y su frecuencia de aparición es dependiente de la aplicación compartida y sería poco eficiente mantener en memoria imágenes de cursores que tal vez no se desplieguen.

## 3.3 Reducción de tasa de bits

Transmitir la interfaz gráfica usuaria como secuencia de imágenes capturadas toma demasiado ancho de anda como video. Actualmente estamos probando mecanismos mejores para distribuir este video sintético. Al menos se ven dos acercamientos. Uno busca definir un muestreo de cuadros de imagen basado en la frecuencia de cambio de estos. Así una región sin cambios en algún intervalo de tiempo puede ser muestreada con menor frecuencia. Los cuadros que cambian más frecuentemente pueden muestrearse con mayor frecuencia, esta alternativa ya se encuentra en etapa de desarrollo, próximamente se obtendrán resultados. También se considera la incorporación de un estándar para la compresión de video tales como H.263+ [17] o H.264 [18], se esperan beneficios de los avances tecnológicos involucrados en estos estándares.

# 4 Resultados a la fecha de las pruebas.

Las mejoras y nuevas características han sido probadas en Windows, Linux y Solaris. Los receptores funcionan bien en todas estas plataformas; sin embargo, la distribución de información del mouse fue implementada y probada sólo en Windows 98, 2000, y XP. No sólo se ha realizado la remoción de la dependencia JAI, si no también se ha comprobado que el nuevo paquete probado funciona mucho mejor. El nuevo codificador JPEG reduce el tiempo de compresión en un 75% y el decodificador acorta el tiempo de descompresión en un 50%. Las pruebas fueron realizadas en una maquina Pentium 4 corriendo Windows 2000, pero no hay motivos para esperar resultados totalmente diferentes en otras plataformas.

Después de la codificación PNG de la figura del mouse en la pantalla, se obtiene una imagen de 500-byte. Esta cantidad es comparable con los 900 bytes de una imagen rectangular (tile) de tamaño promedio después de una compresión JPEG. La utilización del procesador debido al procesamiento del mouse termina por ser muy pequeño (menos de un 2%). La forma del puntero del mouse no cambia frecuentemente, pero su posición si lo hace, por lo que se opta por muestrearla cada 200 [ms]. Este parámetro es configurable. Con estos valores el usuario remoto obtiene una vista del mouse bastante suave sobre la aplicación compartida. Finalmente, debido a que los usuarios pueden iniciar una sesión en cualquier momento, se ha configurado la retransmisión de la imagen del cursor cada 15 [s]. En donde este parámetro también es configurable.

# 5 Conclusiones y trabajos futuros.

La colaboración sincrónica en Internet requiere un servicio para compartir datos además de audio y posiblemente canales de video. Odust es una herramienta distribuida que soporta intercambio de datos mandando la ventana desplegada en la pantalla para cualquier aplicación. El usuario remoto también puede pedir el control y operar la aplicación compartida. Con el objeto de eliminar la dependencia de paquetes externos de Java, se ha cambiado la implementación para la codificación y decodificación de imágenes a Java 2D. Esta nueva implementación ofrece las mismas características que las previas pero a una mayor rapidez. Este cambio también facilita la instalación y uso de Odust.

La transmisión de la figura y posición del mouse realmente mejora el conocimiento y el uso de la herramienta. Esta característica prácticamente no afecta la disponibilidad del procesador ni la utilización del ancho de banda. También se ha notado un menor uso del audio o canales de mensajería instantánea debido a que el movimiento y ubicación del mouse transporta importante información.

En el futuro, se desarrollará una función nativa de captura del mouse para Linux. Actualmente se trabaja en la documentación de Odust con UML y en la revisión del diseño inicial, con el fin de mejorar su modularidad y reutilidad. Usar TCP como protocolo de transporte subyacente también es un trabajo en progreso. Puesto que el receptor se basa completamente en Java y una vez que TCP lo soporte, se planea ver el camino para hacer una versión Applet de Odust. Finalmente, se considera que un cambio en el mecanismo para la transmisión de la

secuencia de imágenes, se espera disminuir considerablemente el ancho de banda y el tiempo de procesamiento de captura. Se cree que un gran aumento en el rendimiento puede conseguirse con el uso de codecs de video estándar como lo supone el esquema estudiado para la transmisión de ventanas de aplicación. La incorporación de nuevos servicios de middleware, particularmente un módulo de audio para que los participantes puedan comunicarse sin necesidad de aplicaciones complementarias, también se considera como trabajo futuro.

## 6 Agradecimientos.

Agradecemos el soporte prestado por el "Instituto Internacional para la Innovación Empresarial" [19] y el proyecto de investigación UTFSM N° 23.04.26.

## 7 Referencias

[1] Weblog, What makes a weblog a weblog?
http://blogs.law.harvard.edu/whatMakesAWeblogAWeblog/
[2] ODUst, http://www.3ie.cl/~odust/
[3] VIC, http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/
[4] RAT, http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/
[5] Messenger, http://messenger.msn.com/
[6] VNC, http://www.uk.research.att.com/archive/vnc/index.html
[7] RealNetwoks, http://www.realnetworks.com/industries/index.html
[8] JAI http://java.sun.com/products/java-media/jai/index.jsp
[9] Agustín J. González, "A semantic-based middleware for multimedia collaborative applications", Ph.D. Thesis, Computer Science Department, Old Dominion University, USA, 2000.
[10] Sun Microsystems, Java language, J2SE http://www.java.sun.com/
[11] JPEG, G.K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol.34, no.4, pp. 30-44, April 1991.
[12] JNI http://java.sun.com/j2se/1.4.2/docs/guide/jni/
[13] C++ Win32, Windows API, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winprog/winprog/windows_api_start_page.asp
[14] MSDN, http://msdn.microsoft.com/
[15] PNG, T. Boutell, "PNG (Portable Network Graphics) Specification: Version 1.0," Request for Comments RFC 2083, January 1997.
[16] Java Image I/O API, http://java.sun.com/products/java-media/imageio.html
[17] H.263+, G. Côté, B. Erol, M. Gallant, and F. Kossentini, "H.263+: Video Coding al Low Bit Rate," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 849-866, November 1998.
[18] H.264, "The emerging H.264/AVC standard", R Schäfer, Thomas Wiegand and Heiko Schwarz, Heinrich Hertz Institut.
[19] Instituto Internacional para la Innovación Empresarial, http://www.3ie.cl/