UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INTEGRACIÓN TRANSPARENTE ENTRE TELÉFONOS INTELIGENTES Y LCDS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

CHRISTIAN EGON BERKHOFF DOSSOW

PROFESOR GUÍA:

SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISIÓN:

JOSÉ ALBERTO PINO URTUBIA

PATRICIO INOSTROZA FAJARDIN

SANTIAGO DE CHILE

SEPTIEMBRE 2010

RESUMEN

Los dispositivos móviles, en particular los teléfonos inteligentes (conocidos como Smartphones), serán nuestro principal medio para interactuar con un futuro mundo digital ubicuo. Las razones que impulsan esta aseveración nacen de sus características innatas: una elevada conectividad, un cómodo tamaño para transportarlos y una creciente potencia computacional. Lamentablemente, una de sus grandes ventajas, el tamaño, es a su vez una desventaja, debido a que estos dispositivos sólo pueden tener una pantalla de reducidas dimensiones. Este es el problema que este trabajo de memoria pretende solucionar.

El objetivo principal consiste en investigar y desarrollar un prototipo, que permita proyectar inalámbricamente la imagen en la pantalla del teléfono inteligente a un televisor LCD de grandes dimensiones. Con esto se busca mejorar la legibilidad y posibilidades de interacción con los contenidos que se encuentran almacenados en el dispositivo. El escenario bajo el cual se visualiza este proyecto, es una sala de reuniones con muchos usuarios turnándose el uso del LCD.

En términos generales la aplicación debe ser sumamente simple e intuitiva de utilizar. Por otro lado, la solución propuesta debe ofrecer además un buen desempeño y calidad a la hora de proyectar las imágenes. Debe existir un compromiso entre las reducidas características de Hardware del Smartphone, las tasas de transferencia de la red inalámbrica, y una imagen que permita seguir fluidamente las acciones y eventualmente leer contenidos.

El proyecto fue desarrollado satisfactoriamente cumpliendo con los todos requisitos establecidos. Para esto se utilizaron las tecnologías y herramientas de desarrollo de Microsoft, así como otros trabajos relacionados. Una generosa porción de esta memoria se dedica a documentar el cómo este proyecto fue llevado a cabo, justificando las decisiones de diseño tomadas.

Tras probar la aplicación y ver sus capacidades, debilidades y principalmente potencialidades, se concluyó que la solución provista es únicamente un primer paso en el ámbito de la proyección de la pantalla. En particular, entregar a las aplicaciones la capacidad de reaccionar ante la proyección, para que modifiquen su interfaz, sería un interesante trabajo futuro. Si bien podría desarrollarse una serie de aplicaciones, que hagan uso de esta capacidad de forma independiente, lo óptimo sería dejar este factor común en el Sistema Operativo del teléfono. Adicionalmente, como proyectos a futuro se propone el investigar el uso de otros dispositivos (como notebooks) y agregar otras funcionalidades a la solución, como por ejemplo controles de rotación de la imagen.

AGRADECIMIENTOS

Parto agradeciendo a mis Padres quienes, a pesar de la distancia y conflictos pasados, me impulsaron y permitieron llegar hasta donde estoy ahora.

En segundo lugar agradezco a mis amigos(as), quienes permanentemente me respaldaron y han estado para aconsejarme o ayudarme.

Finalmente quiero dar las gracias a Don Sergio Ochoa, quien me sugirió este interesante tema de memoria y fue un excelente y confiable profesor guía.

Este trabajo de memoria ha sido parcialmente financiado por el proyecto LACCIR No. R0308LAC005.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE ALGORITMOS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Día a día el uso de tecnologías móviles se está volviendo más cotidiano, esto debido en parte a la creciente necesidad de estar informado de las últimas novedades y conectado a las diferentes redes sociales (Facebook, Twitter, Correo Electrónico, etc.). Los dispositivos móviles permiten un acceso expedito y ubicuo a las tecnologías de la información, en particular los teléfonos inteligentes tienen la capacidad computacional suficiente y conectividad necesarias para acceder a la nube. Además, sus reducidas dimensiones permiten transportarlos con facilidad, lo que permite un acceso fácil a los recursos compartidos.

Claramente el precio de estos equipos es un factor importante, es lo que principalmente define lo masivo que serán estos equipos y en la medida de que sean más populares, habrá aún más interés por extender sus capacidades o de publicar contenidos dedicados. En los últimos años el precio de los equipos de alta movilidad ha disminuido drásticamente, por ejemplo el año 2009 el mercado Chileno fue arremetido por una ola de Netbooks[1]. Los teléfonos inteligentes (iPhone, Blackberry, Samsung Omnia, etc.) se ven como actores aún lejanos, pero en la medida de que sus precios disminuyan, o que aparezcan alternativas económicas, se irán transformando en una opción para los consumidores.

El número de usuarios de teléfonos inteligentes (más comúnmente conocidos como Smartphones) va en creciente aumento. Por citar una cifra; según el reporte de Febrero [1] de la prestigiosa organización comScore, el número de usuarios de teléfonos inteligentes aumentó aproximadamente en un 21%, solamente entre los periodos septiembre-noviembre 2009 a diciembre-febrero 2010.

Muchas empresas han notado la importancia que tomarán los celulares (y la conectividad a Internet en general) en el futuro. Es por esto que gigantes como Google, Microsoft, Nokia, HP, y Samsung están creando Sistemas Operativos y Herramientas de Desarrollo, diseñados para extraer el máximo de las potencialidades de estos equipos. Su objetivo es proveer a los usuarios de un medio propicio para transportar, acceder y compartir la información.

1.1 Problema a resolver

Lamentablemente la gran ventaja de los handhelds (PDAs y smartphones), su tamaño, es a su vez una desventaja. La pequeña pantalla que incluyen es capaz de cubrir (al menos parcialmente) únicamente las necesidades de un usuario que quiera utilizar el dispositivo individualmente, pero es poco práctica o engorrosa si se desea mostrar dicha pantalla a un grupo. La información no es de tanta

[1] Un Netbook es esencialmente un Notebook de menores capacidades y tamaño que generalmente incluye la mayor cantidad de opciones de conectividad (Wifi, Bluetooth, 3G, etc.) posible, para suplir sus eventuales falencias.

utilidad si es que está confinada únicamente al disco duro del handheld, se necesita de una forma apropiada para exponerla a una audiencia.

En una reunión, donde 4 ó 5 personas se juntan para discutir un tema puntual, se podría necesitar de información que está en un handheld. Si lo que se busca es discutir sobre gráficos complejos la reunión será seguramente un fracaso. En esta situación convendría "proyectar" la información a un LCD, proyector u otro dispositivo similar que tenga una pantalla de mayores dimensiones y que permita a todos los espectadores ser partícipes.

Normalmente esto implica tener que conectar o desconectar cables, lo cual hace perder tiempo, y probablemente interrumpe demasiado el proceso de la reunión. Además, los usuarios deben contar con los adaptadores adecuados para poder proyectar la pantalla del handheld (rara vez traen una salida de video estándar). Si tomamos en cuenta la diversidad de estos dispositivos, entonces la complejidad de este punto se acentúa. Además, el hecho de tener el dispositivo conectado a través de un cable a un proyector o LCD, hace que el mismo no sea fácil de mover o compartir entre los miembros de la reunión.

La idea central de este trabajo de título, es proveer de una arquitectura de Hardware y Software que permita mostrar los contenidos que viajan en los dispositivos móviles en pantallas o displays de gran tamaño (Como un monitor dispuesto en una sala de reuniones).

Por ejemplo, los médicos de un hospital podrían llevar las fichas de sus pacientes dentro de sus PDAs. Luego un médico podría querer discutir junto a sus colegas el estado de salud de un paciente. Sería conveniente que la ficha electrónica pudiese mostrarse en un monitor de tamaño apropiado, para que todos puedan verla sin problemas. Adicionalmente, otros médicos podrían desear mostrar las fichas de sus pacientes, por lo que resultaría apropiado poder compartir este "recurso" de LCD entre quienes participan en la reunión.

Este sistema o arquitectura debe ser amigable, sumamente fácil de operar. La integración entre el Smartphone y el display debe ser tan transparente y automática como sea posible. La alternativa propuesta debe entregar una ayuda y no más problemas.

1.2 Relevancia, motivaciones y desafíos

En la actualidad los problemas grandes no son abordados por individuos de forma aislada, sino que son resueltos por equipos de especialistas. Las reuniones y la organización del grupo adquieren enorme importancia y no son una problemática menor. Es necesario proveer las herramientas adecuadas y las nuevas tecnologías entregan oportunidades que no deberían dejarse de lado. En particular este trabajo combina el mundo de los dispositivos móviles, que está ganando enorme popularidad y terreno, con los medios visuales presentes normalmente en una sala de reuniones (monitores, proyectores, etc.).

Desde el punto de vista técnico existían una serie de desafíos que debieron ser afrontados para obtener un proyecto a la altura de lo que se buscaba. A continuación se presenta una enumeración con los distintos desafíos que fueron encontrados.

I. **Hardware:** El primer punto analizado fueron los equipos o componentes a utilizar. Evidentemente se escogieron equipos compatibles con la arquitectura propuesta. Además, la idea fue seleccionar una alternativa que sea viable y realista de un punto de vista tanto técnico como monetario.

II. **Comunicación:** Otra interrogante fue la comunicación entre los diversos equipos. Se tuvo que proveer una arquitectura de red que permitiera la entrada y salida espontánea de usuarios y que sea sumamente simple de configurar. Idealmente no debiesen haber configuraciones de por medio. Por otro lado, la codificación de la información gráfica presentó uno de los mayores desafíos (si es que no fue el mayor). Esta compresión debe ser un compromiso entre uso de la red inalámbrica, la memoria, el procesador del Smartphone, una tasa de refresco aceptable e imágenes de calidad.

III. **Restricciones de bibliotecas:** El desarrollo de una aplicación para dispositivos móviles supone de forma intrínseca un desafío. Además de las restricciones de hardware, existen limitantes para el Software, en particular, si se quiere tener un elevado desempeño (para la compresión por ejemplo), es necesario recurrir a las primitivas de bajo nivel que ofrece el Sistema Operativo. En temas de eficiencia, el desarrollador se encuentra sometido a lo que ofrecen las APIs nativas. En general, implementar funcionalidades por otras vías deriva en código órdenes de magnitud más lento o bien extremadamente complejo. El tamaño de la pantalla del teléfono, la potencia y la diversidad del hardware fueron otra complejidad presente.

IV. **Interfaz con el usuario:** Una gran complejidad nace del aspecto menos riguroso, ya que esta aplicación requiere de una interfaz gráfica intuitiva, simple y que permita un enlace prácticamente automático.

V. **Testing:** A fin de asegurar un comportamiento razonablemente bueno, se realizaron numerosas pruebas del tipo caja negra. Además, se probó el funcionamiento en la mayor cantidad posible de equipos disponibles para asegurar un correcto funcionamiento.

1.3 Objetivos de la memoria

El principal objetivo de esta memoria es diseñar e implementar un software que permita, en forma simple y transparente para el usuario, proyectar la pantalla de un teléfono inteligente en un LCD. Los objetivos específicos que se derivan del objetivo principal son los siguientes:

- Definir el protocolo de interacción entre celulares y LCDs, vía un intermediario si fuese necesario.

- Diseñar e implementar las aplicaciones cliente y servidor que permitan la integración transparente, con su correspondiente interfaz gráfica.

- Desarrollar un método de compresión de la información gráfica, optimizado para la transmisión por redes inalámbricas a tiempo real. Es importante notar que no se considera como objetivo la transmisión de video. Se espera tener una buena calidad de imagen y una tasa de refresco aceptable para notar cambios únicamente.

Estos objetivos son abarcados detalladamente en la sección 3.

1.4 Sobre la organización y convenciones de la memoria

En esta última subsección, tras haber dado una pincelada del problema que se busca resolver, se explica la organización y una convención que se seguirá en este documento.

A grandes rasgos primero se describe y luego formaliza la solución que se está buscando. Después se describe como fue implementado el proyecto en cuestión, partiendo de lo más general y luego dando otros detalles relevantes. Finalmente se concluye basándose en las experiencias adquiridas a lo largo del desarrollo. Siendo más específico, se explicitan las secciones y su intención a continuación:

1. **Introducción:** Recapitulando, esta primera sección introductoria entregó nociones de lo que se busca implementar y ambiente en el cual esto se ve inmerso.
2. **Trabajos relacionados:** En esta sección se comentan trabajos realizados por otros investigadores en la misma línea.
3. **Requisitos de la solución:** Aquí se formalizan las funcionalidades que debe cumplir el proyecto.
4. **Diseño de la solución:** Se entrega una visión global sobre cómo funciona la solución desarrollada.
5. **Diseño detallado del software:** Aquí se detalla la solución en bajo nivel, entrando en aspectos de programación. Es la documentación de menor nivel antes del código fuente mismo.
6. **Implementación de la solución:** Esta sección explica cómo usar la solución desde la perspectiva de un usuario.
7. **Resultados obtenidos:** Aquí se fundamentan algunas decisiones de diseño de acuerdo a los resultados de experimentos de rendimiento realizados.
8. **Conclusiones y trabajo a futuro:** Se concluye en base a lo aprendido y se proponen ideas futuras que podrían ser implementadas en continuaciones de este proyecto.

Finalmente resta por informar que al nombrar elementos técnicos que son propios de la aplicación desarrollada (como nombres de clases, métodos o variables), estos son destacados con *letra cursiva* para distinguirlos. Se hace esto para entregar una ayuda visual al lector, que le indique la aparición de un término nuevo, propio de esta memoria.

2 TRABAJOS RELACIONADOS

Los LCDs, en general todo tipo de pantallas y proyectores, se han vuelto muy comunes en las salas de reuniones. Hoy en día resultan extrañas las reuniones que no sean apoyadas por medios audiovisuales. Es más, los LCDs de gran tamaño (actualmente unas 32 a 40 pulgadas) han llegado de forma abundante a los hogares de consumidores casuales.

Fuera de la sala de reuniones también se han dispuesto pantallas gigantes. Por ejemplo, en espacios públicos como se señala en [2], donde no solo se muestran contenidos sino que además interactúan con los transeúntes. El objetivo de esta pantalla es entretener a los clientes de una empresa de telecomunicaciones mientras esperan su turno, con encabezados de noticias, animaciones y juegos. Investigadores han estudiado de qué formas el uso de estos monitores públicos podría resultar útil. En particular se ha propuesto utilizarlos para publicar informaciones para la comunidad [3] [4] y como un anexo o ayuda a las conversaciones [5] [6].

La elevada frecuencia con que nos encontramos y relacionamos con estos medios y su fuerte vínculo con las tecnologías de la información, incitan a aventurarse a creer que en el futuro la presencia digital será realmente ubicua. Los handhelds son los candidatos a volverse el medio de entrada para interactuar con todos los dispositivos digitales que nos rodearán, entre ellos las pantallas. A modo de analogía, los handhelds pasarían a ser controles remotos universales.

Para interactuar con los LCDs se han sugerido varias estrategias. En [7] enumeran una serie de técnicas que podrían ser utilizadas para interactuar con objetos (como fotografías) que estén siendo mostrados en la pantalla. En el artículo [8] estudian la posibilidad de ocupar handhelds en conjunto con dispositivos más tradicionales, como el mouse y el teclado, como medios de entrada. Así por ejemplo, para una presentación se podría ocupar un Smartphone como control para avanzar o retroceder de diapositivas, que además podría almacenar notas y miniaturas, mientras el mouse y el teclado continuarían sus funciones convencionales. Con esto el computador conectado al proyector sigue desempeñando la misma función, solo que es asistido por el Smartphone. Finalmente, en [9] explican aplicaciones que desarrollaron, las cuales corren en un servidor, a su vez conectado a una pantalla. Estas al recibir mensajes simples de un handheld, a través de un SMS, email o de una cuenta de IM, se encargan de realizar acciones. Por ejemplo una aplicación que controle una votación podría recibir simplemente un número que identifique la opción a seleccionar por cada uno de los miembros de la audiencia.

Si bien los casos anteriores detallan aplicaciones que integran teléfonos inteligentes con LCDs, estas difieren de forma medular con el trabajo presentado en esta memoria. En los casos anteriores la aplicación relevante y los contenidos residían en el LCD (con su respectivo servidor) y el handheld es solo un intermediario, el medio para acceder y controlar la pantalla. En este proyecto en cambio los contenidos y las aplicaciones residen en el teléfono inteligente y el LCD es utilizado simplemente como un medio para desplegar la información de forma más adecuada. Este cambio de enfoque permite

acceder a todas las funcionalidades del dispositivo móvil (con sus desventajas) en lugar que solo aquellas permitidas por el servidor. Dado que el número y calidad de aplicaciones disponibles para estos equipos está en crecimiento, se tiene un panorama favorable.

Esencialmente lo que se busca hacer ("duplicar" la pantalla del celular en un LCD) es posible realizarlo por los medios tradicionales recurriendo a algún tipo cable. Hoy en día algunos teléfonos inteligentes proveen esta característica. Sin embargo, lo que aquí se está buscando es establecer un enlace inalámbrico, con todas las ventajas y complejidades (de la perspectiva del desarrollo) que eso conlleva.

Figura 1. Esquema que resume el funcionamiento de WiDi

Un proyecto similar a lo expuesto en este trabajo está siendo desarrollado por la multinacional y ampliamente conocida productora de procesadores Intel. En conjunto con NetGear estas empresas desarrollaron una solución de software y hardware que permite a los notebooks duplicar inalámbricamente su pantalla en un LCD al cual se encuentra adosado un "adaptador"[2]. Esta tecnología es llamada Wireless Display (acortado como WiDi). La Figura 1 resume el funcionamiento de WiDi.

[2] Esencialmente se trata de un computador pequeño.

Wireless Display guarda 2 diferencias fundamentales con el proyecto aquí expuesto:

I. Solo funciona con los nuevos procesadores Intel[3] y con el adaptador NetGear.
II. Únicamente funciona en notebooks.

La primera limitante, además de tener claros objetivos comerciales, probablemente se ve impulsada por el hecho de que dichos procesadores incluyen un codificador de video implementado a nivel de Hardware. Esta característica es de enorme utilidad para este tipo de aplicaciones, ya que acelera enormemente el proceso de compresión.

Actualmente Intel estaría trabajando en ampliar el universo de dispositivos soportados. Se cree que incluirán tablets y dispositivos móviles a la lista, sin embargo la primera limitante se mantendrá por lo que probablemente pasarán un par de años antes de que se vea un teléfono inteligente con esta tecnología. Como se verá más adelante, la arquitectura usada por WiDi es muy similar a la usada en este trabajo (es igual si se mira en un nivel macro).

[3] Línea Core del año 2010

3 REQUISITOS DE LA SOLUCIÓN

Si bien a lo largo de este documento se ha ido detallando de forma esporádica lo que se espera del proyecto, recién en esta sección se detallarán de forma precisa y formal los requisitos que debe cumplir la solución como para considerarse satisfactoria.

3.1 Funcionalidad básica

Este requisito establece la funcionalidad mínima esperada. Esto es duplicar la pantalla del equipo cliente, transmitirla al servidor y que este la despliegue en su propia pantalla. Más detalles de lo esperado de esta arquitectura cliente-servidor son descritos en 3.2.

En la primera etapa es necesario copiar todo el contenido de la pantalla. No solo el formulario específico de la aplicación que este en primer plano. La idea es incluir las barras de estado y menús, ya que dichos espacios generalmente condensan mucha información útil. Además, sin estos sería difícil que la audiencia pueda seguir y entender las acciones que está realizando un usuario en el cliente.

Lamentablemente hacer esto obliga a tener que recurrir al Sistema Operativo y, dado que este normalmente es reducido, no siempre se cuenta con los medios deseados. Afortunadamente sí existían primitivas que permitían acceder al dispositivo gráfico.

Casos en que las aplicaciones utilicen otras secciones de memoria, distintas a las asignadas por el Sistema Operativo para mantener la información gráfica, son muy difíciles de controlar. Para cada aplicación especial habría que tener una consideración y dado que se espera soportar un universo amplio de dispositivos, soportar todas las aplicaciones sería un objetivo inalcanzable y es por esto que estos reducidos casos no serán contemplados como requisito. Conviene hacer notar que la gran mayoría de las aplicaciones efectivamente utilizan los canales tradicionales para manejar su interfaz gráfica.

En la segunda etapa, correspondiente a la transmisión y en el contexto de este primer requisito, no es un tema relevante el cómo se realice. Este tema es abordado nuevamente en la sección 3.4.

Finalmente, en la tercera etapa, la imagen transmitida es mostrada en la pantalla del LCD. Para que la imagen sea adecuadamente desplegada en el servidor es necesario expandirla a fin de efectivamente aprovechar las dimensiones del LCD. Para evitar distracciones, el resto de la pantalla es pintada color negro. Los botones propios de la aplicación servidor deberán esconderse tras un tiempo de inactividad de los dispositivos de entrada (mouse y teclado). Además, para la interfaz gráfica del servidor se contempla un panel que describa el estado del servidor, enlace y nombre del cliente que se encuentra enlazado. Este panel debiese desvanecerse (al igual que los botones) tras unos segundos para no ocupar espacio de forma innecesaria y mantener la atención del público en lo relevante.

3.2 Arquitectura

Los elementos de Hardware claves en este trabajo son el teléfono inteligente y el LCD. Estos son el requerimiento explícito. Además, y debido a lo poco accesible que es programar un LCD, se adosa un computador a este, que pasa a ser un intermediario entre ambos equipos (de forma similar a como funciona la tecnología Wireless Display descrita anteriormente).

Eventualmente a futuro podría existir un LCD que tenga mayor apertura a aplicaciones de terceros, sin embargo la opción de conectar un computador personal a una televisión o proyector se está volviendo bastante popular y no supone un problema. Ya son numerosas las ofertas de HTPCs (Home Theater Personal Computer) en el mercado y la tendencia va a la alza. Sobre la arquitectura de Hardware, la componente de Software se manifiesta con una aplicación en el dispositivo cliente y otra contraparte servidor en el computador que se encuentra adosado al LCD.

3.3 Restricciones

Al iniciar este trabajo se planteó la posibilidad de trabajar utilizando las tecnologías del gigante Microsoft. Sumado a esto cabe mencionar que se disponían de una serie de trabajos (memorias de grado y magister) previos que hacían uso de dichas tecnologías y que podrían ser incluidas para simplificar el trabajo realizado o que bien podrían ser combinadas a futuro para tener funcionalidades más extensas.

Dado que esta posibilidad no planteaba problema (sino al contrario) y que además se disponía de una librería que simplificaba el manejo de la red (ver 4.2.1) se optó por seguir esta vía. Esencialmente esto no es una restricción, sino que más bien corresponde a una decisión de diseño.

3.4 Conexión inalámbrica

Este requerimiento establece que toda comunicación entre clientes y servidores debe realizarse de forma inalámbrica. Dado que en la actualidad un gran número de teléfonos inteligentes cuenta con la posibilidad de conectarse a redes WiFi se optó por utilizar estas, pues tienen la ventaja de ser ampliamente conocidas y permitir una elevada tasa de transferencia. Esto será de gran ayuda a fin de transmitir la información gráfica de la pantalla del handheld.

El entorno al cual esta aplicación estaría enfocada es uno en que hay numerosos Smartphones y estos entran y salen espontáneamente de la red sin mayor aviso, podría decirse que hay un elevado movimiento. Este escenario debe ser considerado en la aplicación (a nivel de red y de Software) y es por esto que se prefiere el esquema de una red MANET antes que el de una red tradicional.

3.5 Simpleza

Un requisito importante, para el éxito de esta aplicación y distinguirla de otras opciones, tiene que ver con la simpleza. Se tiene que ofrecer una interfaz gráfica intuitiva que permita en pocos pasos establecer el enlace con el LCD. Todo esto sin dejar de lado informaciones relevantes (estado de la red, nombre de los monitores disponibles, etc.). Además, la configuración de la red inalámbrica debe ser automática, a fin de ahorrar a los usuarios inexpertos detalles técnicos de poca relevancia para ellos.

3.6 No interrumpir

La aplicación que se ejecuta en el cliente no debe interrumpir otras aplicaciones que se estén ejecutando. Esto implica que la interfaz gráfica, además de simple, debe ser discreta y capaz de "esconderse". Por otro lado, la aplicación debe evitar, en la medida de lo posible, consumir recursos importantes del sistema como el acceso a disco, procesador y memoria ram.

Dado que la red será configurada por este sistema de forma exclusiva para el enlace, no se tienen requisitos en cuanto al uso de la red. La aplicación podría hacer uso de todo el ancho de banda, siempre que no moleste a otros usuarios que ocupen esta misma red. Como una medida de amigabilidad con el usuario, la aplicación deberá notificar al usuario de los cambios en la configuración de su conexión WiFi.

3.7 Multi-usuario

Si bien la aplicación podría ser diseñada considerando un solo usuario, esto sería alejarse de la situación real en la cual se visualiza este proyecto. Inicialmente esta herramienta fue concebida como una ayuda en la sala de reuniones, es por esto que es imprescindible que varios usuarios puedan interactuar simultáneamente.

Esto implica que la solución debe considerar la presencia de múltiples Smartphones y que cada uno podría potencialmente tener la aplicación cliente instalada. Además, en una sala de reuniones podrían haber dispuestos varios LCDs (cada uno con su servidor), los cuales podrían ser accedidos simultáneamente por clientes distintos.

Actualmente no es un requisito que un servidor maneje más de un enlace simultáneamente, mostrando más pantallas al mismo tiempo, ya que la idea es que el uso de este "Recurso LCD" se vaya turnando.

3.8 Rendimiento

Este último requerimiento es uno de los que establece mayor desafío. Si bien hasta ahora la aplicación descrita a través de los requerimientos previamente mencionados "funcionaría", para tener un resultado realmente exitoso es necesario que cumpla con ciertas características de rendimiento. Esto se traduce en 2 elementos:

i. **Tasa de refresco:** Esto se define como el número de capturas de pantalla del Smartphone (llamados también frames) por segundo que son mostradas por el LCD (FPS). A mayor tasa de refresco, más será lo que los espectadores serán capaces de captar de las acciones que se están realizando. Esencialmente esto permite a la audiencia seguir con fluidez las acciones de quien tiene actualmente el control de la pantalla. Es importante hacer notar que seguir las acciones que se realizan en un Smartphone "Touch" es bastante más difícil que en un Computador Tradicional, ya que no se tiene un cursor que indique donde está la atención de quien está comandando.

   A pesar de las ventajas que trae una elevada tasa de refresco se prefiere que la calidad de la imagen sea mejor. Si la imagen no tiene la nitidez suficiente no servirá de nada, ya que normalmente se busca mostrar algún tipo de texto.

   Dadas las características del hardware, resulta excesivo pedir que las reproducciones de video se vean perfectamente duplicadas. Los requisitos simplemente no contemplan este caso (además correspondería analizar como reproducir simultáneamente el audio). Una tasa de refresco de 4 FPS se considerará aceptable.

ii. **Retraso:** A pesar de las numerosas causas que existen para que la imagen llegue con retraso a la pantalla del LCD[4], debe intentarse de que esto nunca supere un intervalo razonable de tiempo. Un retraso menor a los dos segundos se considera excelente, pero se admiten retrasos mayores en la medida de que estos no sean tan seguidos.

   El problema de un retraso elevado radica en que el usuario del Smartphone puede llegar a estar demasiado adelantado en el tiempo con respecto a lo que ven los espectadores y esto claramente puede derivar en confusiones.

[4]Entre las causas se encuentran el tiempo de adquisición y compresión de la pantalla, búffers en distintos niveles, agrupación de pantallas (enviar cada pantalla independientemente es una mala estrategia para largo plazo) y el retraso natural de la red.

4 DISEÑO DE LA SOLUCIÓN

Una vez claro el objetivo se procede a describir la solución. Esta sección muestra como la implementación realizada cumple con los requisitos establecidos previamente. En esta sección la descripción es de alto nivel, sin entrar en los detalles de diseño y programación. En este sentido una descripción más detallada es entregada en la sección 5.

La arquitectura completa, en particular los programas desarrollados (cliente y servidor), fue bautizada como Clairvoyance (Clarividencia en Francés). Este nombre de fantasía es utilizado a lo largo de este documento para referirse al proyecto en sí. La clarividencia es definida como:

"Capacidad de percepción extrasensorial, por la cual algunas personas parecen recibir información por medios distintos a aquéllos explicables de forma científica. Como su propio nombre indica, esta percepción se caracteriza por captar fenómenos que quedan fuera del alcance de nuestros sentidos, es decir, a través de otros medios" [10].

Sacando los aspectos sobrenaturales de la definición, este término describe satisfactoriamente el principal caso de uso de este Software.

4.1 Arquitectura del ambiente operacional

Lo conveniente es partir revisando la arquitectura de la perspectiva del Hardware (los equipos a utilizar) debido a que son las componentes tangibles y aquellas que han sido comentadas previamente en este documento. La Figura 2 muestra un escenario de ejemplo simple.

Al centro se tiene un computador encargado de coordinar la comunicación. Ha sido llamado Middleware ya que este se posiciona justamente al medio de las componentes que buscan comunicarse (el Smartphone y el LCD). El Middleware, al igual que los Smartphones, utiliza su adaptador de red inalámbrico para crear una red MANET en la medida de que haya equipos en las cercanías. Un solo equipo no puede crear una red MANET, se necesitan al menos dos.

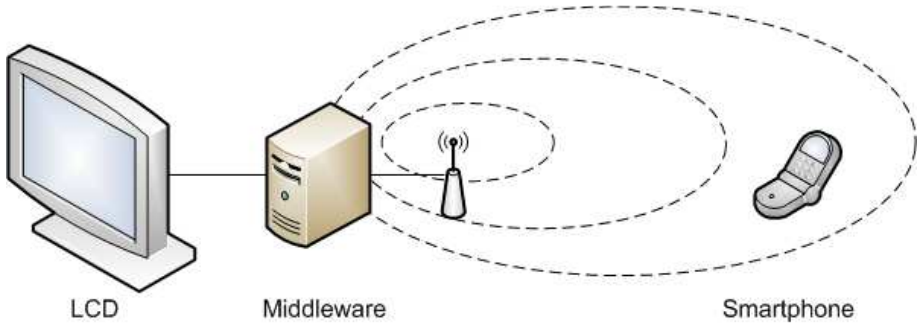


Figura 2. Arquitectura de hardware simplificada

Conectado también al Middleware se encuentra el LCD y es básicamente la pantalla del mismo. Finalmente tenemos el Smartphone que también se encuentra conectado a la MANET y por ende puede comunicarse con el Middleware.

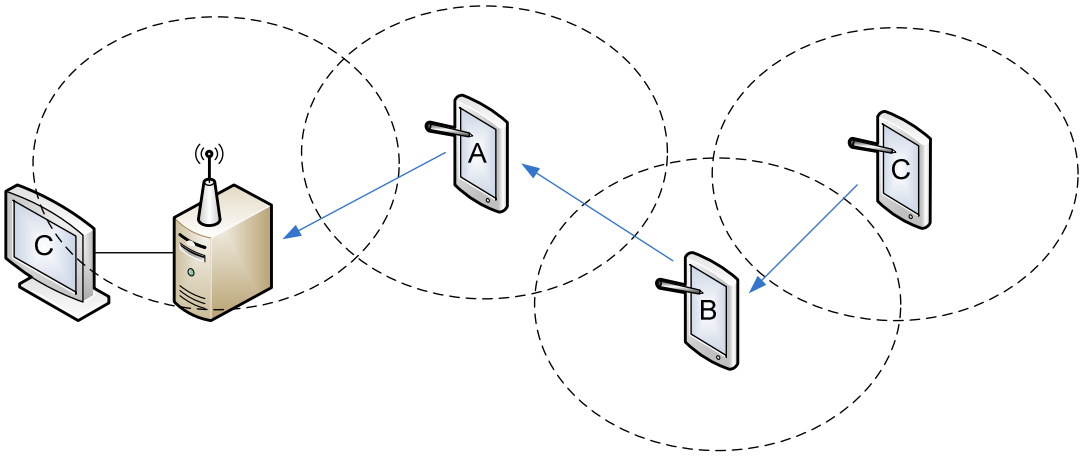


Figura 3. Arquitectura de hardware escenario complejo

La Figura 3 muestra una situación más realista en que se tienen más Smartphones y un solo LCD. El esquema pretende mostrar cómo cada uno de los equipos colabora para crear esta red. En particular, a pesar de que un teléfono este fuera del rango del LCD, otros equipos podrían ayudarle a generar un camino por el cual puede establecer el enlace. Esta capacidad es de utilidad para espacios en los que no existe una infraestructura de red establecida.

4.2 Alternativas analizadas y decisiones de diseño

Como se comentó en la sección 1, existe un amplio abanico de opciones a la hora de desarrollar el sistema descrito. De todas las alternativas se prefirió utilizar la propuesta de Microsoft debido a que existen experiencias y desarrollos previos que hacen uso de sus tecnologías. Las herramientas y APIs de Microsoft tienen bastante madurez en el mercado real y han sido probadas en numerosas ocaciones. Otros trabajos desarrollados sobre las tecnologías Microsoft podrían ser adosados en el futuro.

El lenguaje de programación utilizado para el desarrollo fue C#. La versión de .Net Framework (y .Net Compact Framework para los Smartphones) utilizada fue la 3.5. El ambiente de desarrollo utilizado fue Visual Studio 2008.

4.2.1 Red y comunicación

A fin de implementar esta fundamental característica de forma más simple y acorde a las necesidades de la plataforma se utilizó el Framework[5] de Comunicación HLMP, desarrollado por Juan Francisco Rodríguez como su tesis de magister [11]. HLMP ha sido también documentado en [12] y [13].

Este Framework simplifica la construcción de redes móviles ad hoc (Mobile Ad hoc Network, MANET), el manejo de usuarios de esta red y el envío de mensajes. Sobre este fue necesario implementar el protocolo y el manejo en alto nivel de la comunicación.

Esta librería se encuentra actualmente en desarrollo, aumentando la compatibilidad con distintos equipos y mejorando determinados escenarios. En particular, para el desarrollo de esta memoria se tuvo que mejorar:

- Compatibilidad con Windows Mobile 6.
- Compatibilidad mejorada con los equipos utilizados.
- Mejoras en la velocidad de creación de la MANET cuando se disponía solo de un equipo móvil y un servidor. Este proceso antes solía tardar varios minutos y ahora tan solo toma unos segundos (en promedio).

4.2.2 Smartphone

Los equipos utilizados durante el desarrollo y las pruebas fueron el Samsung Omnia 2 y HTC Touch Diamond 2. Ambos equipos, lanzados en Julio y Abril de 2009 respectivamente, tienen comparativamente con sus pares una capacidad computacional respetable y prácticamente todas las cualidades de un Smartphone actual (GPS, Touchscreen, Cámara Fotográfica y de Video, bastante almacenamiento interno y conectividad a redes Wifi, 3G y Bluetooth). Estos equipos son bastante más poderosos que lo estrictamente necesario para el desarrollo de esta memoria, pero fueron adquiridos con la perspectiva de ocuparlos en proyectos futuros. Estos equipos vienen con el Sistema Operativo Windows Mobile 6.1 Profesional con una interfaz gráfica retocada.

Una de las metas deseables es que el sistema desarrollado funcione sobre el mayor espectro de Smartphones posible. El Microsoft Compact Framework entrega una serie de librerías comunes a todos los equipos Windows Mobile, lo que debiera simplificar esta tarea. Sin embargo existen al menos 3 razones por las cuales, a pesar de que se cuenta con una capa común, Clairvoyance podría no funcionar adecuadamente. Las razones son enunciadas a continuación.

[5] Un Framework es fundamentalmente un conjunto de bibliotecas y herramientas de desarrollo orientadas a simplificar alguna tarea de programación.

4.2.2.1 Bibliotecas nativas

Si bien Clairvoyance fue desarrollado sobre C# y Compact Framework, el acceso a algunos recursos solo se puede hacer utilizando funciones nativas binarias, con el lenguaje de programación C++. Afortunadamente C# provee métodos para acceder a estas funciones por lo que no fue necesario crear un módulo aparte.

El uso de bibliotecas nativas tiene el riesgo de que estas puedan no estar disponibles en todos los equipos. Debido a que son funciones comunes, el riesgo asociado se considera menor, sin embargo corresponde advertir de esta eventual limitante.

Las funciones nativas utilizadas son las que permiten el acceso al adaptador gráfico, copiar la pantalla rápidamente a una región de memoria y comparar 2 regiones de memoria (en particular 2 mapas de bits). Más detalles de estas funciones se pueden encontrar en la sección 5.

4.2.2.2 Requerimientos de hardware

Clairvoyance es una aplicación que hace uso de recursos que no todos los Smartphones o HTPCs pueden poseer. A continuación se enumeran los requerimientos teóricos:

- Servidor
  - Sistema Operativo Windows XP Profesional
  - Procesador de 1.5 Ghz
  - 512 MB de Memoria Ram
  - 1MB de espacio en Disco
  - Tarjeta de Red Inalámbrica
- Cliente
  - Windows Mobile 6.1 o 6.5
  - .Net Compact Framework 3.5
  - Procesador de 500Hmz
  - 200 MB de Memoria Ram
  - 1MB de espacio en Disco
  - Conexión a WiFi
  - Resolución de pantalla: Ha sido probado con 800x480 pixeles, pero debiese ser factible usarlo con resoluciones iguales o superiores a 480x320 pixeles.

Los requerimientos de Memoria y Ram del Smartphone contemplan que el usuario estará haciendo uso de varias aplicaciones simultáneamente (aparte de Clairvoyance).

Clairvoyance fue diseñado para dispositivos cuya resolución de pantalla sea igual o superior a los 800x480 pixeles. Esta fue una decisión de diseño impulsada por el hecho de que al proyectar la pantalla de equipos que tengan una resolución mucho menor, esta se verá muy poco nítida en una pantalla de

grandes resoluciones. Cabe destacar que la mayoría de los equipos actuales tienen resoluciones similares o superiores.

4.2.2.3 "Estándares" no seguidos

Muchos fabricantes de teléfonos inteligentes han alterado la interfaz de Windows Mobile (a fin de entregar una mejor experiencia Touch a sus usuarios) y han creado aplicaciones como reemplazo a aquellas que vienen incluidas de forma nativa. Si estas aplicaciones, o si los cambios a la interfaz no siguen las "reglas" del Sistema Operativo, existe la posibilidad de que Clairvoyance no pueda funcionar adecuadamente.

Esencialmente aquí no se está hablando necesariamente de errores o Bugs, sino que, por diseño, algunas aplicaciones podrían realizar operaciones de distinta manera o, en el peor de los casos, manejar la información gráfica de forma independiente al Sistema Operativo. Por citar un ejemplo, se consideran las aplicaciones para mostrar imágenes tanto del Smartphone Omnia 2 como del HTC.

En primer lugar se analizará el caso Samsung Omnia 2. La Figura 4 muestra el comportamiento de la aplicación por defecto para visualizar imágenes. A la izquierda, a y c muestran el escenario en que la imagen es mostrada a pantalla completa en el celular y esta es exhibida correctamente en el LCD. Por otro lado, a la derecha, se muestra lo que ocurre cuando se voltea el celular. En b se ha girado la imagen para que el usuario del Smartphone la vea en la misma dirección inicial pero deformándola para que entre en pantalla (manteniendo la relación de aspecto). Al proyectar esta imagen (ver d) al LCD la imagen se ve reducida y torcida debido a que los ejes coordenados se han mantenido.

El caso del Smartphone HTC, ilustrado en la Figura 5, es distinto. En primer lugar, al girar el dispositivo, la aplicación no realiza cambios. A la izquierda (a y c) y de igual forma que para el caso del Samsung Omnia 2, la imagen es mostrada de la misma manera tanto en el Smartphone como en el LCD. A la derecha, en b, se ha hecho clic en la imagen y esta es mostrada a pantalla completa. De forma inesperada en el LCD (d) la imagen recibida se muestra rotada. Debido a que tiene un alto más reducido que ancho es posible expandir de mayor manera la imagen, cubriendo casi completamente la pantalla. Esta rotación y traslación del eje de coordenadas es realizada por la aplicación, ya que Clairvoyance **no** rota las imágenes. Simplemente copia una región de memoria en la cual se encuentra la información gráfica. Esta singularidad es particularmente útil para aprovechar el tamaño del LCD.

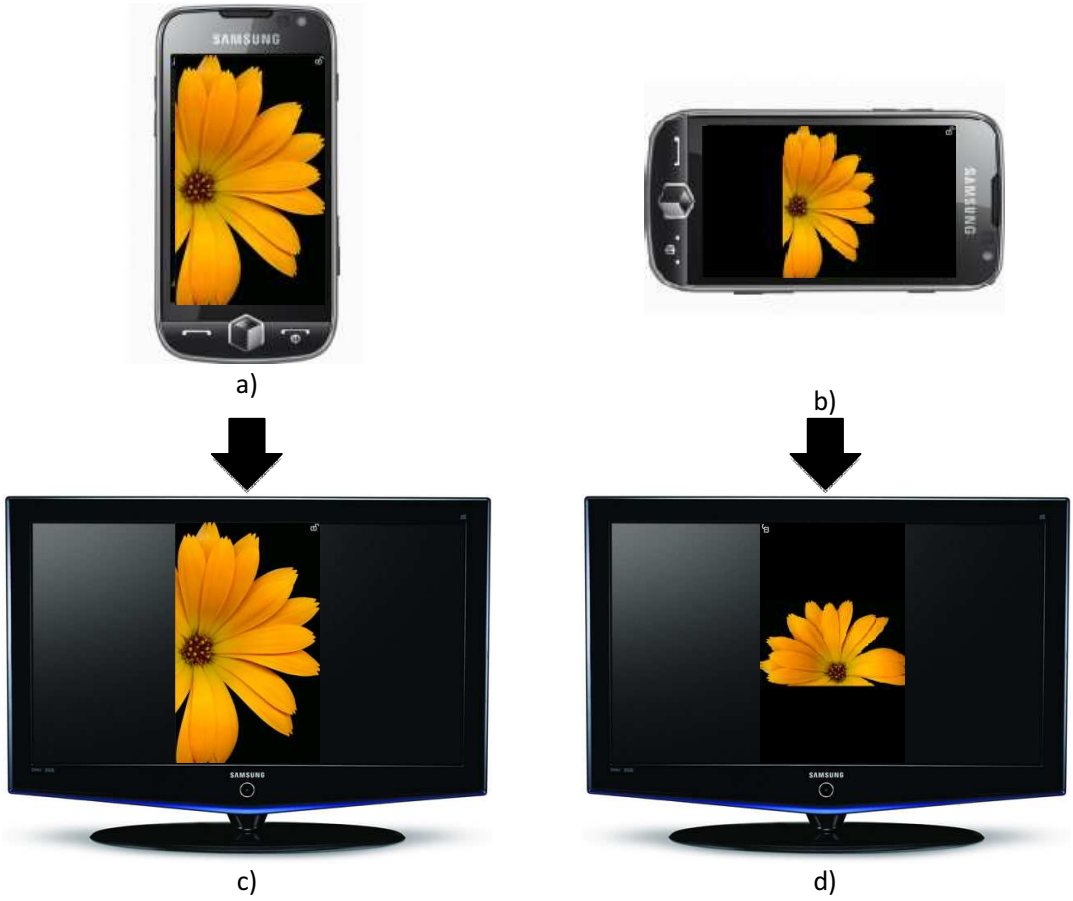


Figura 4. Imágenes en Samsung Omnia 2

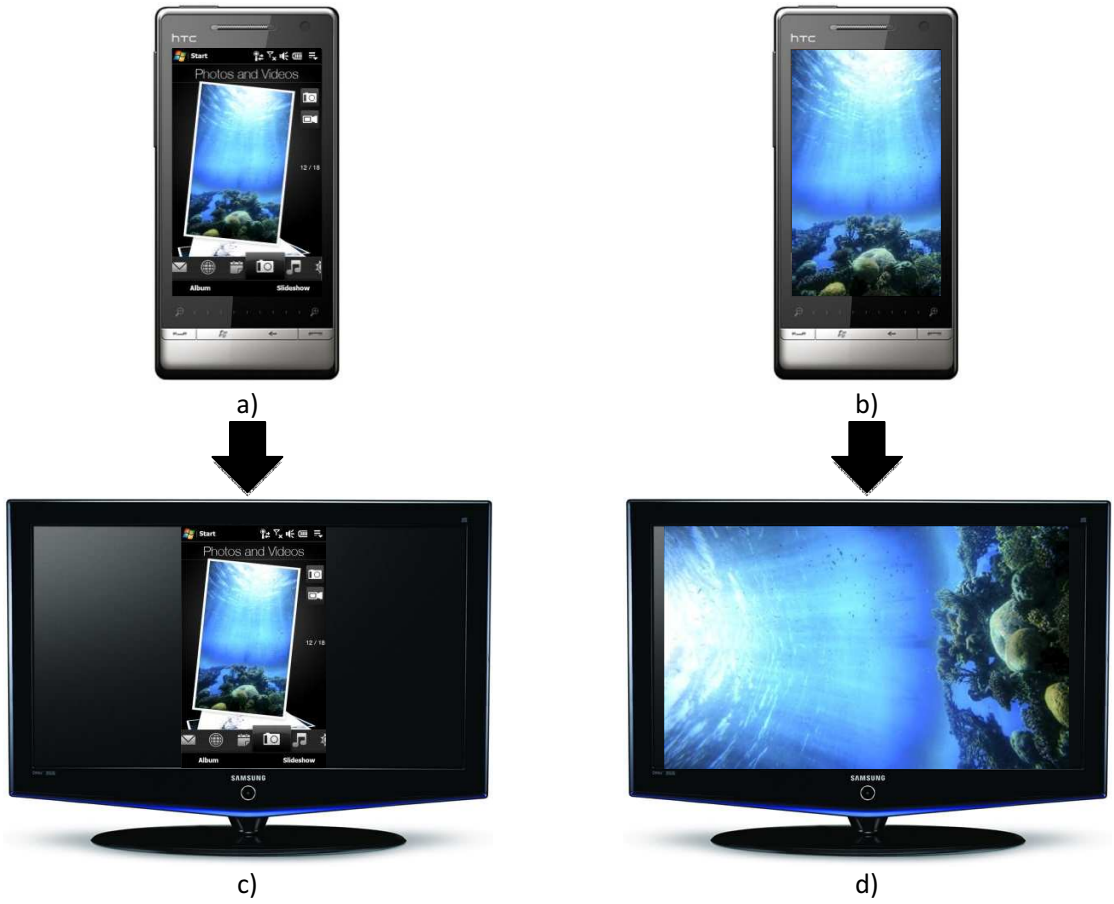


Figura 5. Imágenes en HTC Diamond 2

4.2.3 Middleware

Como Middleware se prefirió un equipo que pueda ser montado detrás de un LCD, para así tener una sola unidad LCD + Middleware que sea fácil de ubicar en una sala de reuniones. Para esto, el gabinete del equipo debe soportar el estándar Vesa que utilizan los soportes traseros de los LCDs (ocupado para montarlos por ejemplo contra las paredes). Adicionalmente, el equipo debe tener un tamaño reducido.

El equipo debe contar con una tarjeta inalámbrica para redes WiFi y puertos para conectar otros dispositivos de entrada. Además, por una restricción de la biblioteca de comunicación HLMP, el equipo

debe utilizar el sistema operativo Windows XP[6]. Es así como se llegó al Nettop[7] MSI Windbox que reúne dichas cualidades. La Figura 6 muestra este equipo y como se monta este detrás de un Monitor LCD.

Figura 6. MSI WindBox detrás de LCD

4.2.4 LCD

Para los fines académicos, prácticamente cualquier LCD del mercado sirve. El único requerimiento es que tenga una entrada compatible con el Middleware (en este caso VGA) y que tenga un tamaño y resolución adecuados.

4.3 Arquitectura de software

Ya teniendo claras las componentes físicas que están en juego, se continúa describiendo cómo éstas colaboran entre sí mediante la capa de software. Esta descripción es breve y tiene como intención enumerar los distintos módulos y procesos que componen la aplicación. La Sección 0 se encarga de entregar una descripción detallada. El software a desarrollar sigue una arquitectura cliente servidor, el cliente corre en los Smartphones y el servidor en el Nettop adosado al LCD.

[6] A la fecha es el único Sistema Operativo para equipos de escritorio soportado por HLMP.

[7] El nombre nace de la idea de un Netbook pero pensado como un computador de escritorio (Desktop).

4.3.1 Estados, mensajes y flujo del cliente y servidor

En una aplicación que sigue la arquitectura Cliente-Servidor claramente hay una gran relación entre ambas componentes. En este caso el servidor es un actor pasivo, que espera las peticiones de los clientes, son ellos quienes deben tomar la iniciativa.

La Figura 7 muestra como ambas partes se relacionan en una ejecución "ideal", es decir, donde no ocurren problemas. Los casos de errores son algo más complejos y por ello son vistos posteriormente al final de esta subsección (dificultan innecesariamente un primer acercamiento). Volviendo a la figura, los círculos indican estados y en particular aquellos con doble borde indican el estado inicial. Los estados agrupados en la mitad superior son aquellos por los cuales pasa el cliente y los inferiores pertenecen al servidor. Las líneas punteadas indican un cambio de estado gatillado por el usuario (por ejemplo al presionar el botón conectar) y las líneas continuas indican que el cambio de estado es generado por un mensaje, el cual es enviado o recibido desde la red. Los tipos de mensaje posibles son *Link*, *Screen* y *Unlink*. A continuación se describe el flujo de la aplicación cliente.

El primer paso consiste en conectarse o crear la MANET. Esto se logra utilizando la librería HLMP, la cual se encarga se configurar el adaptador de red del equipo y de buscar otros equipos que se encuentren en las proximidades. Cuando se encuentra otro equipo se crea una red nueva, pero cuando hay una red ya establecida HLMP se adhiere a esta. Al conectarse los usuarios a la red estos anuncian al resto su llegada y otros datos útiles (como un identificador, un nombre, etc.).

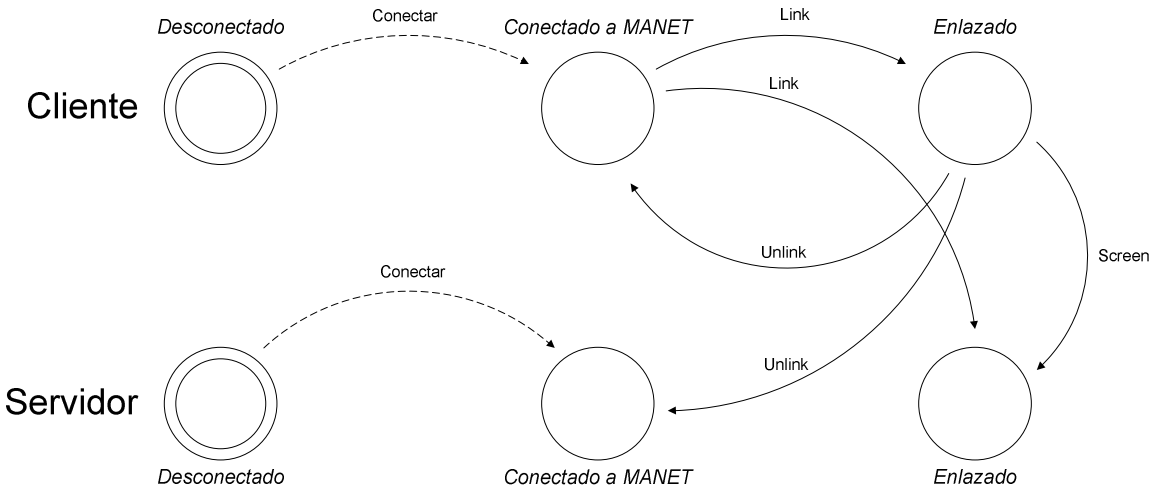


Figura 7. Diagrama de estados y mensajes

Mientras la red esté establecida, HLMP envía mensajes de control constantemente (para saber si los usuarios no se han desconectado, si han cambiado su nombre, etc.). Este mensaje de control fue extendido indicando si es que el usuario de la red es un cliente o servidor y para declarar si está ocupado

o disponible (en total 2 bits adicionales). El mensaje es difundido a todos los usuarios de la red, de este modo los clientes son permanentemente notificados de la presencia de servidores y si estos están disponibles o no.

Cuando ambos equipos están conectados a la red MANET, es el usuario del cliente quien debe dar la orden de establecer el enlace con el servidor de su elección. Para esto se envía un mensaje, más precisamente una petición de *Link*, al servidor. Ésta contiene el tipo de dispositivo (de momento únicamente Smartphones, pero podría incluirse otros en el futuro), la resolución de la pantalla y el tipo de compresión a utilizar. Los métodos de compresión son descritos en 4.3.2. Al recibir esta petición, el servidor evalúa si los parámetros son compatibles (actualmente siempre son compatibles, pero a futuro podría no ser así) y si está disponible (el servidor podría estar atendiendo a otro cliente). Si no existen trabas, el servidor responde afirmativamente y se establece el enlace (ambos pasan al estado Enlazado), en caso contrario el servidor responde negativamente y no se cambia de estado.

Cuando está enlazado, el cliente envía periódicamente las imágenes al servidor utilizando el códec (método/estrategia de compresión) seleccionado. Las imágenes son enviadas en un Mensaje de tipo *Screen*. En estos momentos el usuario puede enviar Clairvoyance al fondo, cambiar de aplicación, y proyectar los contenidos que quiera. Finalmente, cuando el usuario lo desee, puede terminar el enlace, lo que se traduce en el envío de un mensaje del tipo *Unlink*. Con esto ambas aplicaciones pasan al estado "Conectado a la MANET". Ahora otro usuario puede tomar el control del servidor.

Esencialmente existen 4 problemas que se pueden dar en el esquema anterior.

- ***Link* Timeout:** Se da si el servidor nunca responde la petición de link. Se produce un timeout tras 5 segundos. Ni el cliente ni el servidor cambian de estado.
- ***Link* No Aceptado:** Se produce si por alguna razón el servidor no acepta la petición de link. Ni el cliente ni el servidor cambian de estados.
- ***Screen* Timeout:** Si el servidor no recibe un mensaje del tipo *Screen* en 15 segundos se produce un timeout y termina el enlace. Esto vuelve al servidor al estado "Conectado a la MANET".
- ***Screen* No Aceptado:** Si el servidor no acepta el mensaje del tipo *Screen* (porque no está enlazado con este cliente), el cliente vuelve al estado "Conectado a la MANET". Si el servidor recibe el mensaje sin problemas, no se envía una confirmación (solo se envía cuando hay un error). De este modo, si el servidor termina el enlace por *Screen* Timeout, cuando el cliente envíe un mensaje se le responderá que no fue aceptado, luego caerá en este caso y termina el enlace.

4.3.2 Transmisión de información gráfica

A lo largo del desarrollo de Clairvoyance se fueron implementando algoritmos de transferencia y codificación de la información gráfica (llamados estrategias de compresión) cada vez más poderosos y a

la vez complejos. Estos Códecs artesanales están sometidos a las funcionalidades que provee el sistema operativo, por lo que en general no se habla de métodos demasiado sofisticados. Cabe destacar que para la plataforma Windows Mobile 6 no se encontraron librerías que permitiesen realizar encoding de video de forma profesional ni mediante Hardware, lo que motivó el desarrollo de una técnica propia. Si a futuro Clairvoyance fuese desarrollado para otros dispositivos o plataformas, debiese analizarse la disponibilidad de codificadores. Al menos teóricamente, a Clairvoyance se le pueden anexar codificadores de cualquier tipo.

4.3.2.1 Compresión Simple

El primer método de compresión es internamente conocido como "compresión simple". Este captura la imagen en pantalla cada segundo y la compara con una copia obtenida el segundo anterior. Si esta es distinta, entonces se comprime al formato PNG, es empaquetada en un mensaje del tipo *Screen* y enviada al servidor para que la exhiba. En caso de que la pantalla no haya mostrado cambios, se envía de igual manera un mensaje, solo que este indica que no hubo cambios. Esto debe ser realizado para que el servidor no termine el enlace por *Screen* Timeout (ver 4.3.1).

4.3.2.2 Compresión BDS

El segundo método de compresión es el resultado de varias iteraciones de desarrollo. Además es aquel que viene seleccionado por defecto en la aplicación. Fue nombrado Compresión BDS ya que utiliza búffers (B) y adicionalmente, dependiendo de las circunstancias, obtiene capturas de pantallas más pequeñas o a tamaño normal (escalamiento dinámico, dynamic scaling, DS).

Como se aludió, este método tiene 2 calidades de imagen, una que mantiene la resolución y gran parte de la calidad de la imagen (Modo Calidad) y otra que reduce el tamaño y por ende la calidad (Modo Rápido). La Figura 9 muestra una comparación entre ambos modos, cuando se está desplegando en la pantalla una aplicación. De forma similar, la Figura 10 es una comparación a la hora de mostrar una foto. Lamentablemente la calidad de impresión de este documento puede no ser lo suficiente como para distinguir las diferencias. En cualquier caso, más adelante, en la sección 5.4, se detalla en términos objetivos las diferencias entre los modos. Ambas compresiones son realizadas utilizando el formato JPG, las razones de esto son explicadas en la sección 7. La captura de pantalla con el Modo Rápido es aproximadamente 8 veces más rápida que la del Modo Calidad y además utiliza un tercio de la memoria, esto en términos estimativos, ya que depende de numerosos factores. La comparación de dos mapas de bits en el modo rápido también es notoriamente más rápida.

La Figura 8 resume los estados por los cuales pasa esta compresión y será discutida a continuación. Inicialmente este algoritmo utiliza el Modo Rápido. Si la pantalla presenta cambios estos son almacenados en una nueva imagen, pero no son enviados inmediatamente. En esta compresión se acumula un cierto número de capturas de pantallas (o frames) antes de enviarlas al servidor. Si no hay cambios en la pantalla, de forma similar a la compresión simple (descrita en 4.3.2.1), se incluye un frame

falso. Además se tiene un contador (variable t en la Figura 8) que aumenta en uno cada vez que no hay cambios y vuelve a cero cuando ocurren cambios en la pantalla del Smartphone.

Cuando el contador t supera un parámetro M, la compresión pasa al Modo Calidad. En este Modo se captura **una** pantalla que es básicamente la misma que ya se tenía, solo que en este caso con calidad mayor. Inmediatamente después de eso la compresión vuelve al Modo Rápido. Además, internamente la aplicación define una variable de tal modo que no se vuelva a pasar al Modo Calidad para esta misma captura de pantalla (esto para prevenir el caso en que nuevamente pasen M intervalos de tiempo sin cambios en la pantalla).

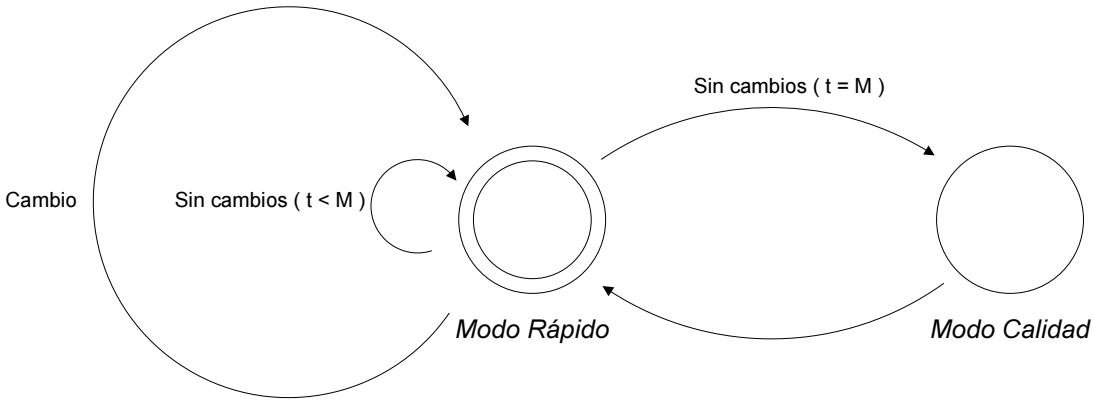


Figura 8. Estados de la compresión BDS

Como se mencionó previamente, esta compresión acumula frames antes de enviarlos para mejorar la tasa de envío global. En general resulta más rápido enviar un mensaje grande antes que varios pequeños. Resumiendo, los frames son de 3 tipos:

i. **Null:** Frame que indica que no ha habido cambios.
ii. **Fast:** Ha habido cambios. Estos son comprimidos en una imagen de baja calidad.
iii. **Quality:** Ha habido cambios. Estos son comprimidos en una imagen de alta calidad.

Este algoritmo de compresión es un compromiso entre una tasa de refresco aceptable y calidad de imagen. Intuitivamente los usuarios que quieran mostrar contenidos que requieran una imagen clara (con texto por ejemplo), esperarán a que transcurra un corto intervalo de tiempo sin realizar acciones. Como por otro lado la tasa de refresco “por defecto” es elevada, permitirá a la audiencia notar los cambios y - como finalidad última - entender lo que está ocurriendo.

Finalmente cabe destacar que los parámetros de esta compresión son modificables y han sido puestos a punta para sacarle el máximo provecho y cumplir con los requerimientos de eficiencia establecidos en 3.8.

Figura 9. Comparación aplicación para Modo Rápido y Calidad

4.3.2.3 Recepción de la información gráfica

Al llegar los mensajes tipo *Screen* al servidor, estos no se muestran inmediatamente en la pantalla, sino que son encolados. Esto se realiza para evitar detener la reproducción innecesariamente por demoras esporádicas en el envío o recepción. El tamaño de la cola no debe ser muy grande para evitar retrasos (ver 3.8).

En la medida de que estas irregularidades se vuelvan comunes o muy extensas (en tiempo), el Servidor Clairvoyance aumenta el tamaño del búffer dinámicamente para intentar mostrar un video fluido. Estos cambios son notificados a los espectadores como se muestra en 6.2.

Antes de mostrar las imágenes recibidas, estas son escaladas para que tengan un tamaño adecuado. En caso de que la aplicación servidor esté en Modo Pantalla Completa, todas las imágenes son

estiradas a fin de aprovechar las dimensiones del LCD. Si se está en Modo Ventana, solo los frames de tipo Fast deben ser escalados (para que tengan el mismo tamaño que un frame tipo calidad).

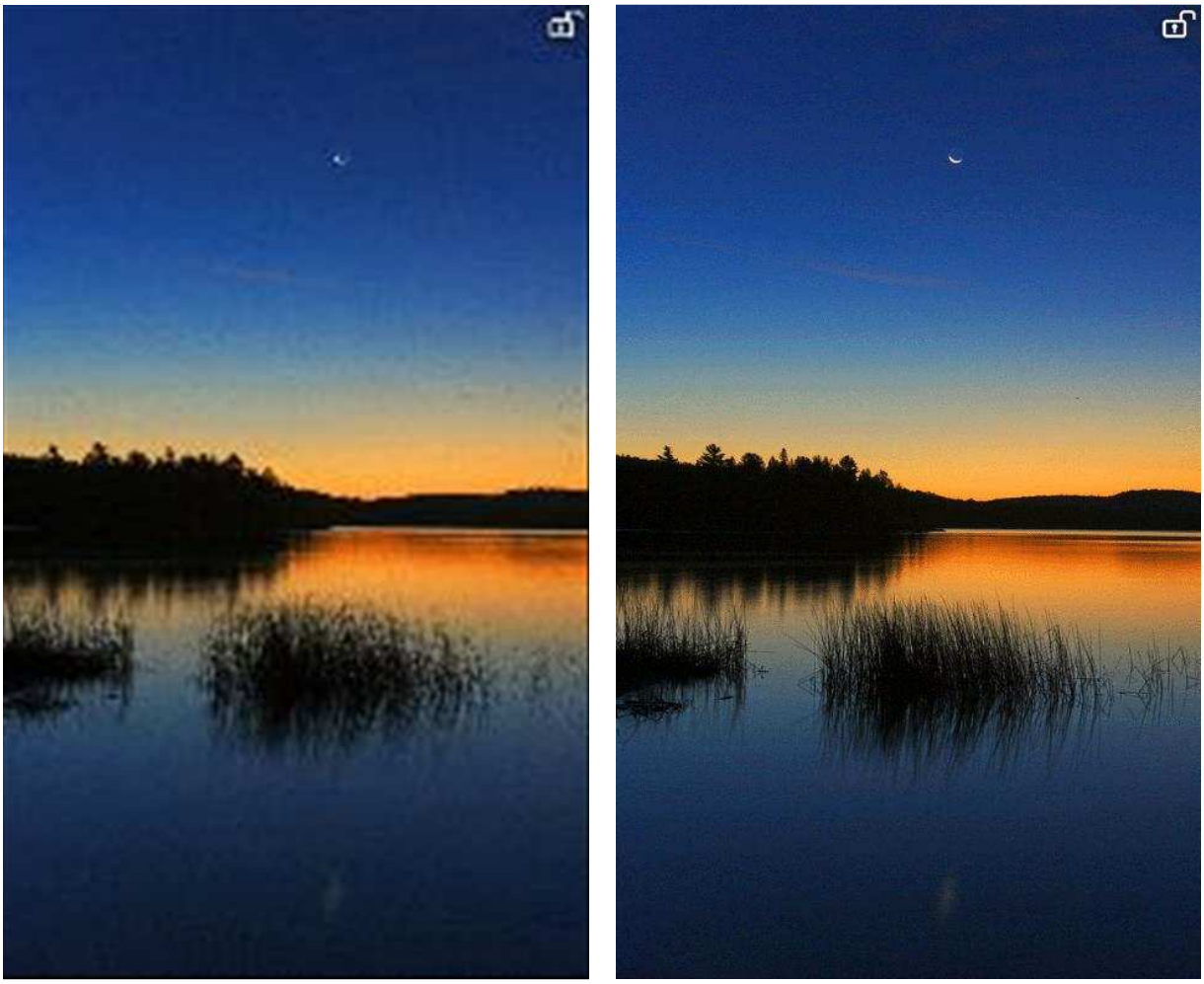


Figura 10. Comparación foto para Modo Rápido y Calidad

5 DISEÑO DETALLADO DEL SOFTWARE

En esta sección se estudia el Software, poniendo énfasis en los detalles que en las secciones anteriores han sido dejados de lado. Se presume un conocimiento a grandes rasgos de las funcionalidades del sistema (en resumidas cuentas tener conocimiento previo o haber leído los capítulos anteriores de este documento). La concentración estará ahora puesta en aspectos técnicos y de programación.

5.1 Organización del proyecto

La primera división importante en el Software nace de su arquitectura. Se tienen dos componentes; Cliente y Servidor. Ambas partes fueron tratadas como proyectos aparte.

Existen dos aspectos similares (que tienen clases en común) en ambos proyectos. Estas son la Comunicación y la Codificación. Claramente ambas partes tienen que hablar un "idioma común" para entenderse y por otro lado el servidor debe ser capaz de Reconstruir la información que le envía el cliente.

El Framework HLMP, utilizado para la comunicación, está compuesto de dos librerías. Una para dispositivos móviles y otra para equipos de escritorio (con el sistema operativo Windows XP). Esta diferencia se debe a que esencialmente se tiene que desarrollar para ambientes dispares. Para dispositivos móviles se tiene acceso a un conjunto bastante menor de funcionalidades y el hardware también es más limitado. Contradictoriamente, este esquema de dos librerías aparte, refuerza la idea de tener dos componentes de comunicación completamente separadas (sin clases en común). Por otro lado, independiente de las diferencias internas, HLMP muestra al desarrollador una interfaz idéntica tanto para el caso de los dispositivos móviles como para computadores de escritorio. Esta clara dicotomía fue afrontada utilizando el preprocesador de C# (C# Preprocessor) y la posibilidad de incluir un archivo por enlace[8] que ofrece Visual Studio 2008. Dentro de ambos proyectos se definió un símbolo condicional de compilación[9] único (PocketPC para el cliente y WINXPSP3 para el servidor). Utilizando estas variables se delimitaron las áreas que pertenecían exclusivamente al cliente o al servidor. Cuando el preprocesador lee el código fuente, dependiendo del símbolo condicional de compilación vigente, eliminará las partes que no corresponden. De la perspectiva del compilador será como que dichas partes simplemente existiesen. Esto fue utilizado con frecuencia para que una clase común a ambos proyectos

[8] Al incluir un archivo de este modo, los cambios que se realicen en cualquiera de los 2 proyectos alterarán el archivo. El efecto es idéntico al de un link simbólico en UNIX.

[9] Conditional Compilation Symbol. Esencialmente un parámetro que es entregado en tiempo de compilación.

referenciase a la versión apropiada para su dispositivo de HLMP y para declararse dentro del espacio de nombres (namespace) adecuado.

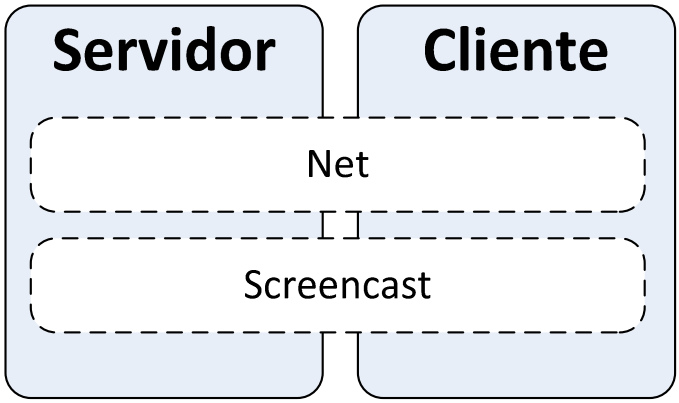


Figura 11. Arquitectura de software

Recapitulando, la Figura 11 muestra la arquitectura de Clairvoyance de la perspectiva del Software. Como se mencionó, la primera división nace de las componentes de Hardware, separando al cliente y servidor. Dentro de cada una de estas componentes se encuentran elementos propios de cada uno, por ejemplo la interfaz gráfica. Además se tienen 2 aspectos altamente relacionados que son la comunicación y la codificación, las cuales tienen el espacio de nombres "*Net*" y "*Screencast*" respectivamente. Además de código fuente, cada proyecto tiene asociado otros recursos como por ejemplo imágenes e íconos.

A continuación se detallará cada una de las componentes mencionadas: Servidor, Cliente, *Net* y *Screencast*. Los esquemas y en general las descripciones que se darán solo hacen mención a lo importante, dejando casos rebuscados de lado (en general los casos de error). Es una simplificación, cuya extensión debe bordear aproximadamente la mitad de lo que realmente existe. Esto se hace para no oscurecer la documentación con detalles, que aunque importantes para la aplicación en un nivel de producción, no son relevantes para dar a entender cómo se realizan las tareas. Para una descripción total se recomienda revisar el código fuente.

5.2 Servidor

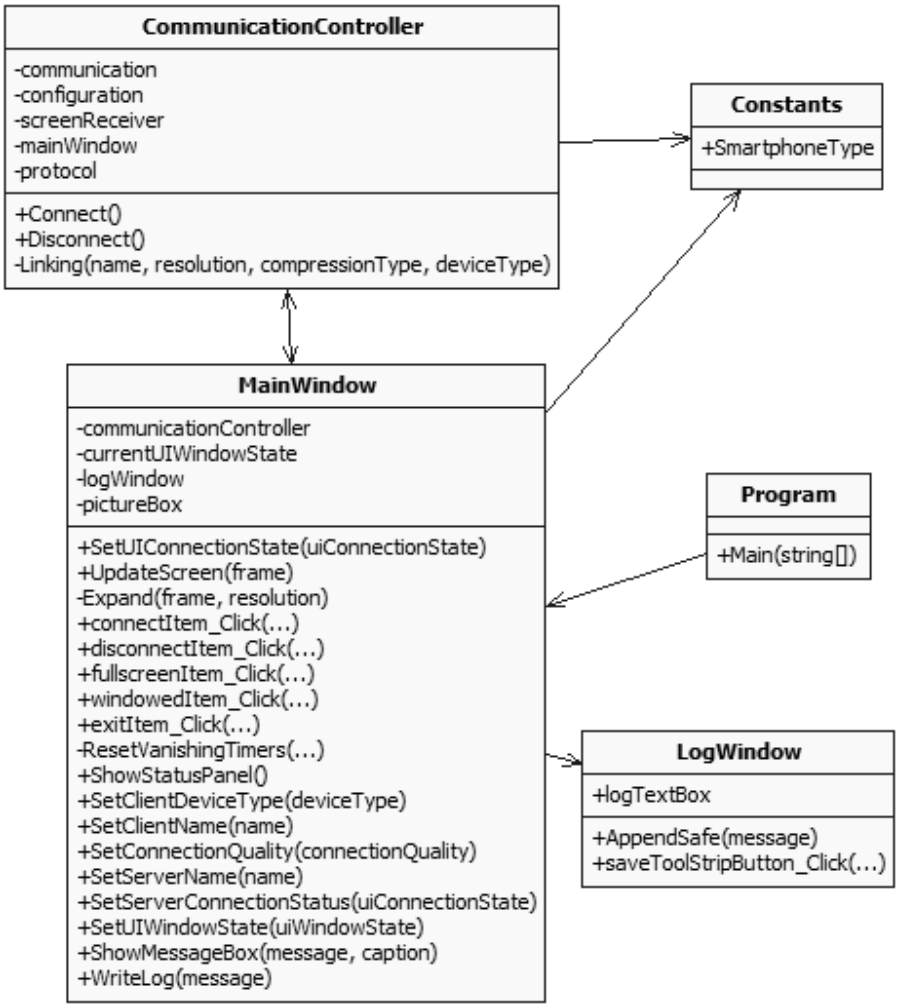


Figura 12. Diagrama de clases exclusivo del servidor

Las clases exclusivas del servidor son principalmente dos: *CommunicationController* y *MainWindow*. *Program* es una clase autogenerada la cual se encarga de iniciar la aplicación. *Constants* es una clase compartida tanto por el cliente como por el servidor la cual contiene identificadores para los dispositivos clientes soportados por el sistema, en este caso, solo Smartphones. Finalmente *LogWindow* es una clase que representa una ventana con la bitácora de eventos (o Log) del Servidor de Clairvoyance. Provee un método para anexar un mensaje y otro para guardar toda la bitácora en un archivo.

MainWindow, como su nombre lo indica, es la ventana principal de la aplicación servidor. Tiene dos modos de funcionamiento: a pantalla completa y como ventana. El modo es cambiado utilizando

SetUIWindowState (quien a su vez define la variable *currentUIWindowState*). Esta interfaz está compuesta principalmente de:

i. Un área en la cual se exhibe el frame recibido del cliente, llamada *pictureBox*. El método *UpdateScreen* se encarga de actualizar esta imagen expandiéndola si es necesario usando el método auxiliar *Expand*.
ii. Una barra con botones que permiten entre otras cosas: conectarse (*connectItem_Click*), desconectarse (*disconnectItem_Click*), pasar a modo pantalla completa (*fullscreenItem_Click*) o modo ventana (*windowedItem_Click*) y salir (*exitItem_Click*). Este panel se desvanece tras unos segundos de inactividad del mouse.
iii. Un panel que detalla el estado de Clairvoyance. Este panel provee métodos públicos que son utilizados por *CommunicationController* a fin de notificar de cambios. Este panel muestra el nombre (*SetServerName*) y estado (*SetServerConnectionStatus*) del servidor, el nombre (*SetClientName*) y tipo (*SetClientDeviceType*) de cliente conectado y la calidad de la conexión (*SetConnectionQuality*). Este panel también se desvanece tras segundos de inactividad pero reaparece si es que hay algún cambio de estado.

Finalmente, el método *SetUIConnectionState* se encarga de actualizar toda la interfaz del servidor a fin de que represente el estado de la conexión y enlace.

CommunicationController es el real protagonista del lado del servidor. Esta clase se encarga de configurar la Red, establecer el enlace, mantener el protocolo, delegar el trabajo de efectivamente recibir los frames del cliente y además tiene el deber de actualizar la interfaz gráfica con los últimos eventos.

5.3 Cliente

El cliente guarda muchas similitudes con el servidor. Acá también son dos clases las primordiales: *MainForm* y *CommunicationController*. Ambas cumplen un rol similar a sus versiones del servidor. También se tiene un puntero a *Constants* y una clase *Program* que únicamente se encarga de iniciar la aplicación.

La interfaz gráfica del cliente (Smartphone) es descrita en la clase *MainForm*. En términos simples esta consta de un botón para conectarse (*actionButton*), otro para terminar el enlace (*linkEndButton*), una lista de íconos con los servidores disponibles y un botón para salir. Para evitar que el Smartphone se suspenda (para ahorrar uso de la batería) se invoca periódicamente el método *systemIdleTimer_Tick*, que le indica al Sistema Operativo que se está trabajando. Normalmente, mientras se usa Clairvoyance, el usuario deja el Smartphone tranquilo unos minutos pero no desea que el enlace se pierda. Adicionalmente a lo descrito, en *MainForm* se encuentran bastantes elementos irrelevantes de la perspectiva de los objetivos de este capítulo.

CommunicationController además de configurar y establecer el canal de comunicación utilizando el protocolo, tiene clases auxiliares que le asisten en las tareas de establecer el enlace con un servidor y emitir las capturas de pantalla. Cuando un nuevo usuario entra en la red este es inspeccionado utilizando el método *CheckUser*. Si es un servidor y se encuentra disponible, este es añadido a la lista ocupando el método público *AddDisplay* de *MainWindow*.

Finalmente, dentro del cliente se tiene una clase *Tests* utilizada para realizar pruebas de velocidad y rendimiento a las distintas estrategias de compresión probadas. Como esto no es un parte propia de la aplicación, sino que un elemento auxiliar, no fue incluida en el diagrama.

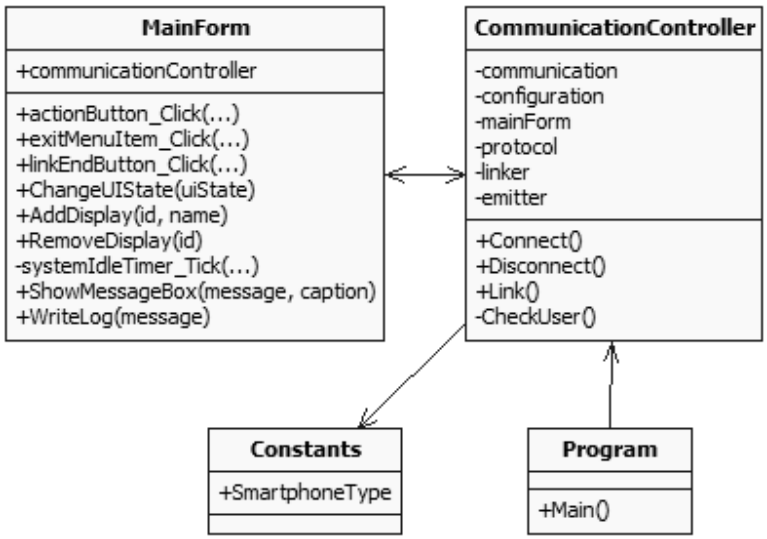


Figura 13. Diagrama de clases exclusivo del cliente

5.4 Screencast

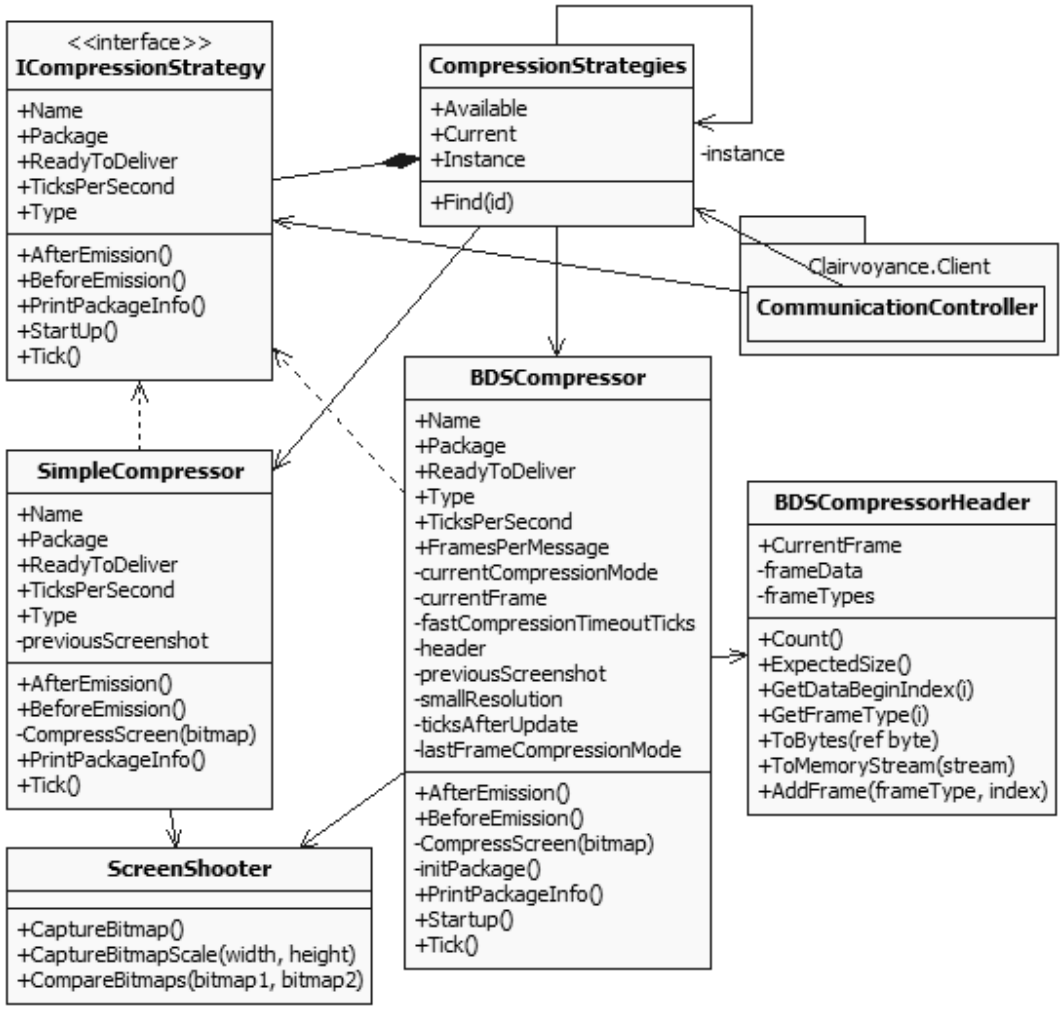


Figura 14. Diagrama de clases de Screencast del cliente

Esta componente[10] de Clairvoyance consta de 2 partes. La primera, encargada de la compresión y empaquetamiento de la pantalla del cliente, es resumida en el diagrama de clases de la Figura 14. La segunda parte, encargada de desempaquetar la información enviada, es resumida en el diagrama de clases de la Figura 16. Como es de esperarse, ambas componentes tienen un alto grado de dependencia y es por esto que son descritas en conjunto en esta subsección.

[10] En términos de bajo nivel esto corresponde más precisamente a un espacio de nombres (namespace).

El Algoritmo 1 muestra como las estrategias de compresión son utilizadas, a través de la interfaz *ICompressionStrategy*. En este pseudo-código se han ignorado por simpleza casos de error que sí son considerados por la aplicación. Además la función *Wait* está implementada de tal forma que empieza a contar el transcurso del tiempo cuando se inicia el ciclo, no cuando es invocada. De este modo, si realizar las operaciones dentro del ciclo toma 30 milisegundos y el argumento de *Wait* son 70 milisegundos, entonces este esperará 40 milisegundos antes de continuar.

```
cs := CompressionStrategies.Instance.Current
cs.Startup()
While( linkAlive )
            cs.Tick()
            If ( cs.ReadyToDeliver )
                cs.BeforeEmission()
                Log ( cs.PrintPackageInfo() )
                Send ( cs.Package )
                cs.AfterEmission()
            Wait ( 1000 / cs.TicksPerSecond )
```

Algoritmo 1. Uso de las estrategias de compresión

El primer paso consiste en obtener la estrategia de compresión que se tenga actualmente seleccionada, la cual se adquiere directamente de la única instancia (por seguir el patrón de programación Singleton) de *CompressionStrategies*. Después se invoca al método *Startup* a fin de que la estrategia tenga tiempo de realizar preparaciones previas al ciclo. Luego se entra a un ciclo que perdurará hasta que se pierda el enlace con el servidor. En este ciclo lo primero que se hace es invocar el método *Tick*, ideado para que la estrategia realice el conjunto de operaciones relacionadas con la periódica captura y compresión de la pantalla. El paso siguiente consiste en averiguar si la estrategia de compresión tiene el paquete listo para ser enviado, utilizando la propiedad *ReadyToDeliver*. Si esto es efectivo se imprime en la bitácora (*Log*) de Clairvoyance información descriptiva del paquete y luego es enviado al servidor con *Send*. Además, a la estrategia se le entrega la posibilidad de reaccionar previa y posteriormente con los métodos *BeforeEmission* y *AfterEmission* respectivamente. El último paso es esperar unos cuantos milisegundos y luego el ciclo se reinicia.

Se tienen además 2 clases que representan estrategias de compresión: *SimpleCompressor* y *BDSCompressor* la cual por su parte hace uso de *BDSCompressorHeader*. Durante el proceso de desarrollo existieron otras estrategias, pero estas clases se unificaron y ahora conforman *BDSCompressor*.

ScreenShooter es una clase auxiliar que provee métodos estáticos (no es necesario crear una instancia de *ScreenShooter*) para capturar la pantalla del celular ya sea a tamaño completo o en una cierta escala. Para realizar esto se utilizan las funciones nativas del sistema operativo *GetDC*, *BitBlt* y *StretchBlt* las cuales se encuentran definidas en la biblioteca nativa *coredll.dll*. Además *ScreenShooter*

provee de una función para comparar dos mapas de bits. Esto se realiza de forma rápida ocupando la función *memcmp*[11], que también proviene de *coredll.dll*.

El proceso de compresión ya fue brevemente discutido en 4.3.2. Aquí se busca mostrar como dicho proceso es llevado a cabo ocupando las clases mencionadas. Dado que la compresión BDS posee todas las cualidades de la simple sólo se describirá la primera. A continuación se explicita como cada característica fue implementada.

i. **Modo Rápido y Modo Calidad:** *BDSCompressor* funciona básicamente siempre bajo el Modo Rápido. La variable *ticksAfterUpdate* aumenta en uno cada tick (llamada a la función *Tick*) en que no hay diferencias en la pantalla. Cuando su valor supera el parámetro *fastCompressionTimeoutTicks* correspondería agregar una imagen de alta calidad. Antes hacerlo se revisa que la variable *lastFrameCompressionMode* no corresponda al modo calidad. Esto para evitar enviar la misma imagen dos veces (esto ya que dentro de *previousScreenshot* solo almacena imágenes de baja calidad, ver iii)

   El parámetro *smallResolution* define el tamaño de la captura de pantalla para el Modo Rápido. Es el parámetro que es entregado a la función *CaptureBitmapScale* de *Screenshooter*.

   Podría decirse que no existe de forma real el Modo Calidad (no es un Modo), pero resulta más simple explicar esta estrategia imaginando que existen dos modos.

ii. **Buffer:** En vez de enviar cada captura de pantalla de forma independiente a través de la red, conviene, por motivos de eficiencia, reunir un cierto número de frames antes de despacharlos. Para poder unir los arreglos de bits que representan las imágenes es necesario tener un encabezado que establezca entre otras cosas dónde empiezan y dónde terminan los bits correspondientes a una imagen. La Figura 15 ilustra cómo está conformado el paquete generado por el Compresor BDS y que es emitido al servidor.

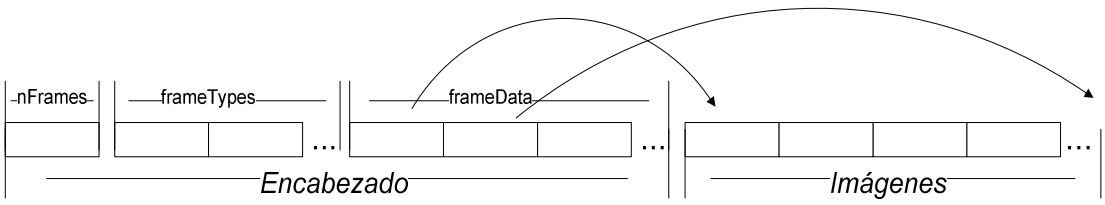


Figura 15. Paquete de compresor BDS

[11] Aparentemente esta función es bastante más rápida que cualquier estrategia implementada en C# (por ejemplo calcular un Hash de ambos mapas de bits y luego compararlos).

El paquete se encuentra dividido en un encabezado y en una región en que se almacena toda la información de las imágenes. Dentro del Encabezado, el primer byte es utilizado para indicar cuantos frames vienen en este paquete (actualmente siempre es la misma cantidad pero a futuro podría tenerse un número variable). Seguido a este viene una tira de bits denominada *frameTypes*. Por cada imagen se asignan 2 bits dentro de este arreglo a fin de identificar de qué tipo es. Por esto *frameTypes* es un arreglo de $\lceil nFrames/4 \rceil$ bytes. Así, por cada imagen, el primer bit indica si es que el frame es nulo o si es una actualización y el segundo indica el modo. Explícitamente:

Bits	Tipo de Frame
00	Frame Nulo
10	Actualización. Imagen de Baja calidad.
11	Actualización. Imagen de Baja calidad.

Tabla 1. Representación binaria de los distintos tipos de frame

Después de *frameTypes* está el arreglo *frameData* que mantiene los índices a las posiciones dentro del paquete que ocupa cada imagen (el índice en cual comienza). Para cada uno de estos se ocupa una variable de tipo entera de 4 bytes. Con esto *frameData* es un arreglo de $nFrames \cdot 4$ bytes.

Para simplificar todo este manejo de bits se creó la clase *BDSCompressorHeader* que tiene métodos que permiten agregar frames (*AddFrame*), conocer de antemano el tamaño del encabezado (*ExpectedSize*), obtener información de un frame en particular (*GetFrameType* y *GetDataBeginIndex*) además de facilitar la creación de su representación binaria (*ToMemoryStream*).

Para la compresión de las imágenes de utilizó el formato Jpg, por ser el más rápido disponible y entregar una calidad aceptable en cualquier escenario.

Justo antes de enviar el paquete al servidor (en el método *beforeEmission*) es anexado el encabezado (que se encuentra en la variable *header*) al paquete. En el método *afterEmission* se reinician variables para el nuevo paquete.

iii. **Sólo actualizaciones:** Esta última característica se refiere a solo enviar capturas de pantalla si es que hay cambios. Para realizar esto se guarda una copia del último frame enviado en la variable *previousScreenshot*. En un siguiente tick se evaluará si es que existen cambios utilizando la función *CompareBitmaps* de *ScreenShooter*.

Dado que las imágenes de alta calidad son generadas después de un tiempo sin cambios, estas no representan nada nuevo. Por esto solo imágenes de baja calidad son

almacenadas para las comparaciones. Dado que la disminución de calidad es causada a su vez por una reducción en la resolución podrían existir cambios que no son notados por la plataforma. Afortunadamente esto no supone un problema ya que un cambio que modifique mínimamente una imagen (en el peor de los casos un par de píxeles) no es de interés y probablemente no será notado por la audiencia. Por un lado se ahorra memoria y procesador del dispositivo cliente y por otro se evita enviar cambios innecesarios.

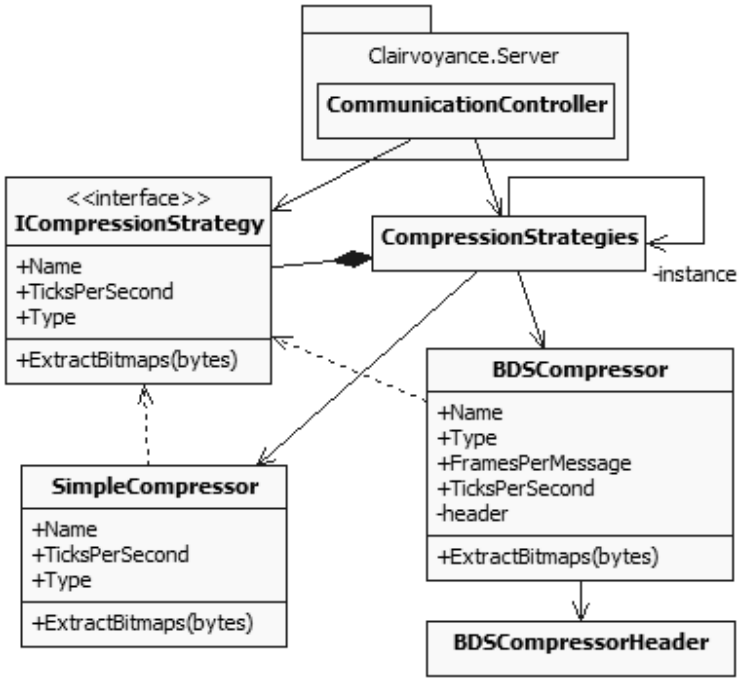


Figura 16. Diagrama de clases de Screencast del servidor

Del lado del servidor el escenario es bastante más simple. Están presentes prácticamente las mismas clases exceptuando a *ScreenShooter*, la cual no tendría sentido acá. Las estrategias de compresión y su interfaz común (*ICompressionStrategy*) solo son utilizadas para almacenar información y se les ha agregado el método *ExtractBitmaps*. *CompressionStrategies* y *BDSCompressorHeader* siguen exactamente igual que en el cliente. Las clases aquí presentes corresponden al mismo archivo que en el cliente, solo que algunas partes han sido removidas o agregadas utilizando el Preprocesador de C# (ver 5.1).

5.5 Net

Siguiendo la pauta utilizada para describir la sub-sección anterior, la Figura 17 muestra el Diagrama de clases para el espacio de nombres Net del cliente. Su contraparte en el servidor es ilustrada en la Figura 18[12]. Si bien ambas partes tienen un alto grado de similitud y comparten muchas clases, sus funciones son distintas. El servidor es un actor pasivo que espera solicitudes y responde. Es el cliente quien debe tomar la iniciativa y enviar las solicitudes de sus requerimientos (de enlace).

Este espacio de nombres es el que mantiene mayor número de relaciones con las distintas componentes. Por un lado provee los mecanismos base de comunicación para la aplicación (que es orquestada por *CommunicationController*) y además debe utilizar las interfaces *ICompressionStrategy* y *CompresssionStrategies* de *Screencast* para construir los mensajes que almacenan la información gráfica (*ScreenMessage*). Finalmente debe implementar interfaces de la librería HLMP para realizar las operaciones de red de bajo nivel. Esencialmente el espacio de nombres Net busca entregar una interfaz simple para que *CommunicationController* administre. Específicamente, las clases con las cuales interactúa y sus funciones son las siguientes:

i. ***UserDataAdmin:*** Como se mencionó previamente, HLMP envía periódicamente mensajes a la red para conocer el estado de los usuarios de la MANET. Este Framework provee además la posibilidad de adjuntar a la información base del usuario (definida en la clase *NetUser* de HLMP) cualquiera otra que se estime necesaria. Para realizar esto debe utilizarse la variable *UpLayerData*, declarada dentro de *NetUser*. Debe tenerse en mente que debe limitarse al máximo el uso de esta variable, ya que al ser enviada constantemente a toda la red, adjuntar demasiada información podría sobrecargar la MANET.

[12] Los atributos y operaciones fueron eliminados en las clases que no presentan cambios para evitar redundancia

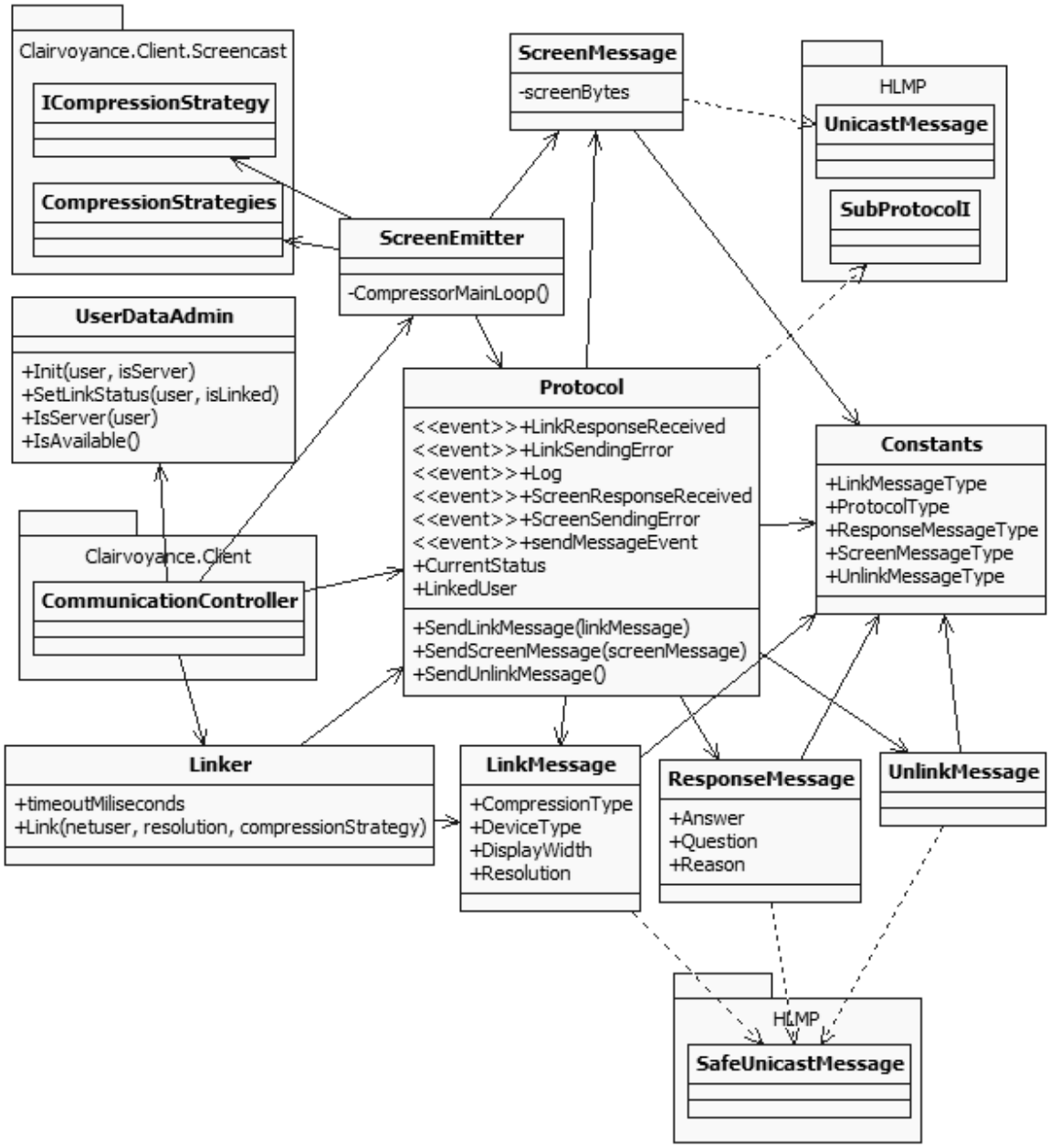


Figura 17. Diagrama de clases de Net del cliente

Afortunadamente son solo 2 variables binarias las que desean darse a conocer al resto de los usuarios de Clairvoyance: Si el usuario es cliente o servidor y si está Disponible o está Ocupado (enlazado). La clase *UserDataAdmin* provee las funciones estáticas *Init* y *SetLinkStatus* para inicializar la variable *UpLayerData* (como cliente o servidor) y para

actualizar el estado del enlace (Enlazado o Disponible) respectivamente. Además provee las funciones *IsServer* e *IsAvailable* para averiguar si un usuario es un servidor o si se encuentra disponible. Todo esto ocupando 2 bits, pero que por limitantes del lenguaje pasan a ser un byte.

ii. ***Linker:*** Simplifica la tarea de establecer el enlace, volviendo este proceso una simple llamada a una función bloqueante *Link*. Esta función retorna un valor booleano que dictamina si se pudo establecer satisfactoriamente el enlace. Bloquea el funcionamiento hasta que el servidor responde o bien hasta que se cancela el enlace por demorarse demasiado (timeout).

Para realizar esta labor, *Linker* hace uso tanto de la clase *Protocol* como de *LinkMessage*.

iii. ***ScreenEmitter:*** Esta clase se encarga de comunicarse con el espacio de nombres *Screencast* y enviar los *ScreenMessages*, que contienen la información gráfica, al servidor. Para realizar esto *ScreenEmiter* ejecuta el método *CompressorMainLoop* que es básicamente el Algoritmo 1 que fue descrito previamente. Este método sigue el patrón Template Method, el cual consiste en ejecutar una serie de métodos de una clase abstracta (la idea es que pueda ser reemplazada por varias clases, de forma similar al patrón *Strategy*). En este caso la instancia es la estrategia de Compresión que se tenga seleccionada.

Al establecerse el enlace, *CommunicationController* crea una nueva instancia de *ScreenEmitter* la cual subsiste hasta que este mismo encuentra un problema (como que el servidor rechace los mensajes tipo *Screen*) o bien hasta que se termina el enlace de forma normal.

ScreenEmitter puede ser visto como la componente de red del espacio de nombres *Screencast*.

iv. ***Protocol:*** Si bien *CommunicationController* no hace mayor uso del protocolo de forma directa, es quien lo construye y guarda la referencia. El único caso en que *CommunicationController* hace uso de la clase *Protocol* de forma directa es para terminar el enlace, ya que es una tarea simple y no hace falta tener una clase adicional para ello.

Tanto *Linker* como *ScreenEmitter* tratan de lidiar de forma independiente con el servidor a fin de simplificar el trabajo de *CommunicationController*, sin embargo existen casos que estos por si solos no son capaces de controlar. Estos casos (principalmente casos de error) son notificados y atendidos mediante el uso de eventos.

Todas las operaciones de red son realizadas por el Protocolo. Es la capa más baja de Clairvoyance y debajo de este empieza el Framework HLMP, cuya descripción claramente escapa los objetivos de esta memoria. El protocolo implementa la interfaz *SubProtocolI*.

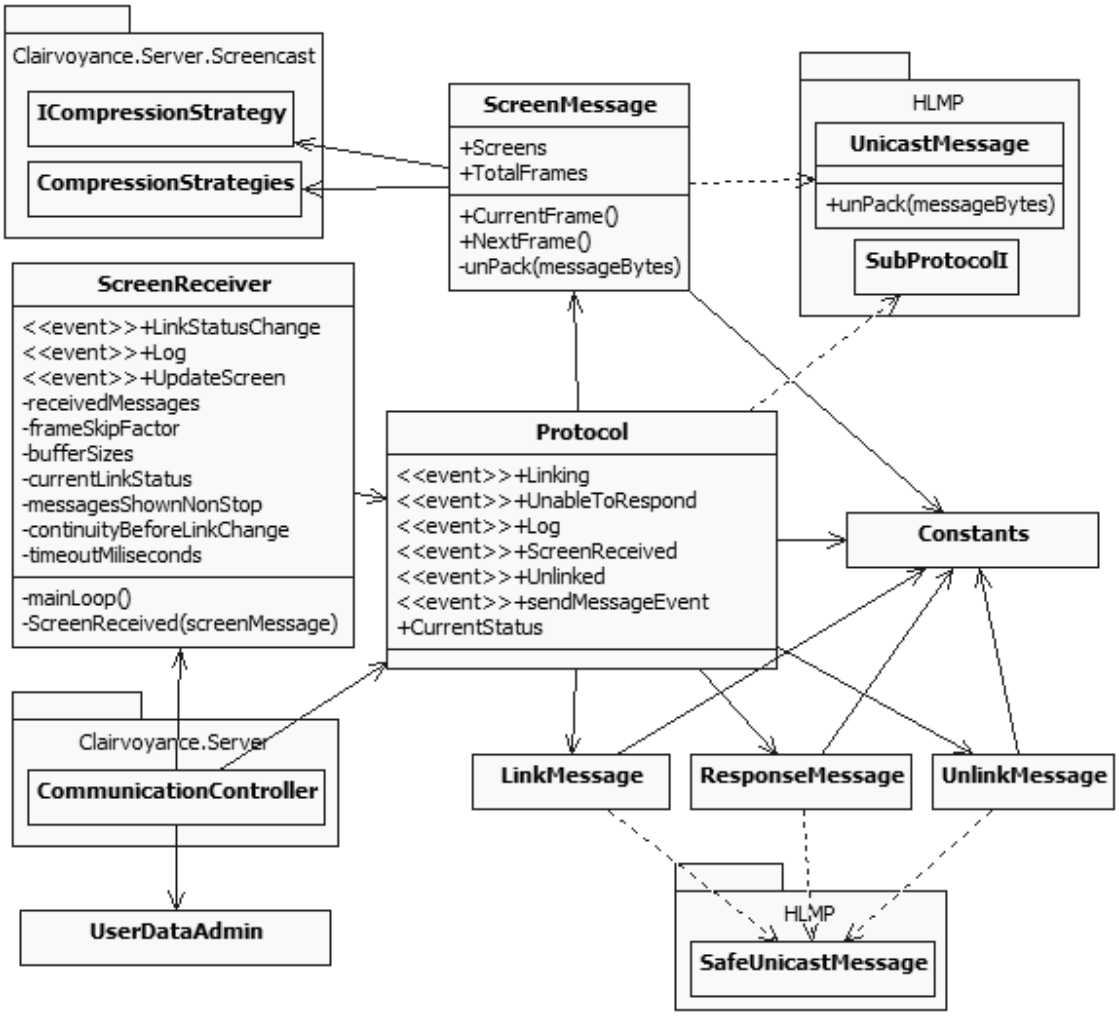


Figura 18. Diagrama de clases de Net del servidor

Los mensajes del tipo *Link*, *Unlink* y *Response* implementan la interfaz *SafeUnicastMessage*, ya que necesitan de la mayor fiabilidad disponible. Por otro lado *ScreenMessage* hereda de *UnicastMessage*, el cual es ligeramente más rápido que *SafeUnicastMessage* pero que es algo más inseguro únicamente en el escenario en que se encuentre muy lejos del servidor y que tenga que ser enrutado, de Smartphone a Smartphone (ver Figura 3), demasiadas veces. *UnicastMessage*, a pesar de ser un mensaje TCP, no asegura que llegue a destino si es que debe viajar a través de demasiados

dispositivos para llegar a su destino. En general se presume que quien intente proyectar su pantalla al LCD intentará naturalmente a su vez acercarse al servidor para mejorar la calidad del enlace.

Aunque existía la posibilidad de utilizar mensajes del tipo *FastUnicast*, que se diferencia por utilizar el protocolo UDP, no fue posible usarlos ya que el tamaño de la información gráfica excedía el tamaño del MTU de la red (limitante de tamaño para los mensajes UDP). Para diferenciar los mensajes, HLMP requiere de identificadores para cada mensaje. Estos se definen en la clase *Constants*.

Las similitudes entre el cliente y el servidor son extensas. Como puede verse en la Figura 18, muchas clases no tienen cambios con respecto a su versión del cliente. En este caso el protocolo está enfocado en recibir información y provee los eventos para dichos casos.

Por otro lado *ScreenMessage* es modificado de tal modo que ahora muestra los mapas de bits de cada uno de los frames (antes simplemente contenía el paquete con la información comprimida y el encabezado). La transformación de arreglo de bits a imágenes es realizada por el método *unPack* utilizando la estrategia de compresión seleccionada.

Además de *UserDataAdmin*, la clase *ScreenReceiver* ayuda a *CommunicationController* en la tarea de hacer funcionar al servidor. Una vez establecido el enlace, *CommunicationController* delega la tarea de recibir las imágenes a *ScreenReceiver*. Este posee los eventos *ScreenUpdate* y *LinkStatusChange* para que la interfaz gráfica pueda ser notificada de los cambios. El primero indica que hay que cambiar la imagen que se está exhibiendo en pantalla y el segundo notifica que la calidad de enlace ha cambiado (esto en base a la cantidad de *ScreenMessages* recibidos y los que se esperaban).

Dentro de *ScreenReceiver*, la variable *timeoutMiliseconds* establece cuanto será el tiempo máximo que se esperará la llegada de un *ScreenMessage*. Si en dicho intervalo de tiempo no llega ninguno, el enlace se termina.

El método *mainLoop* se encarga de ir consumiendo los *ScreenMessage* que se hayan recibido. Estos son almacenados en una cola (*receivedMessages*) al llegar. El método *mainLoop* se mantiene dentro de un ciclo siempre que existan mensajes que consumir. Cuando no hay, se detiene y espera una señal antes de continuar.

Al recibir un nuevo mensaje (evento que es capturado por el método *ScreenReceived*), se evalúa el tamaño de la cola (buffer) de mensajes. Sólo si es que la cola tiene una cantidad de elementos igual o superior al tamaño del buffer esperado (*bufferSizes*) se envía la señal para que *mainLoop* continúe. Se utiliza este enfoque para evitar que *mainLoop* tenga que molestamente esperar con demasiada frecuencia por intervalos cortos de tiempo. La Tabla 2 muestra los valores de la variable *bufferSizes* para sus respectivas llaves.

Así por ejemplo, cuando llega un mensaje, y el estado de la red (*currentLinkStatus*) es *Overloaded*, se enviará a *mainLoop* la señal para que continúe solo si es que el número de mensajes en la cola es igual o mayor a 3.

Llave	Valor
Normal	2
Overloaded	3
Critical	5

Tabla 2. Valores por defecto para *bufferSizes*

El estado de la red se inicializa como *Normal*. En la medida de que se tenga que esperar, porque no quedan mensajes en la cola, la variable *currentLinkStatus* cambiará de valor. Por simpleza el algoritmo funciona de la siguiente forma:

$$te = \text{tiempo de espera}$$

$$cr = \text{calidad de la red}$$

$$bs_{cr} = \text{valor de bufferSizes para una calidad de la red}$$

Con esto, el nuevo estado de la red ($\widetilde{cr}$) pasa a ser:

$$\widetilde{cr}(te) = \begin{cases} \text{Normal}, & te < bs_{Overloaded} - bs_{normal} \\ \text{Overloaded}, & bs_{Overloaded} - bs_{normal} \leq te < bs_{Critical} - bs_{normal} \\ \text{Critical}, & bs_{Critical} - bs_{normal} \leq te \end{cases}$$

Lo anterior resulta más claro con un ejemplo. Si el tiempo de espera fuese de 0.5 segundos entonces, para los valores de *bufferSizes* dados, el estado continuaría siendo *Normal* ($te < 1$), con una cola de 2 mensajes. Si el tiempo de espera fuese de 1.5 segundos el estado pasaría a *Overloaded* ($1 \leq te < 3$). Finalmente si el tiempo de espera fuese mayor o igual a 3 segundos se pasaría a *Critical*.

Por otro lado, si después de que *ScreenReceiver* evaluó el estado de la conexión como *Critical* la red se normaliza y los mensajes llegan con normalidad, entonces se irá paulatinamente "mejorando" la calidad de la conexión y se reducirá el tamaño del buffer. La variable *messagesShownNonStop*, como su nombre lo indica, establece cuantos mensajes han sido procesados por *mainLoop* sin tener que detenerse a esperar la llegada de un *ScreenMessage*. Si este valor iguala al parámetro *continuityBeforeLinkChange* (predefinido como 10) entonces la calidad de la conexión mejora en uno. Si es que estaba en estado *Critical* pasa a *Overloaded* y si estaba *Overloaded* pasa a *Normal*.

Finalmente *ScreenReceiver* también está diseñado para reaccionar si es que el tamaño de la cola es demasiado grande. En este caso el LCD estaría mostrando un elevado retraso con respecto a lo que se

ve en la pantalla del Smartphone. Se eliminará el primer mensaje de la cola hasta que la siguiente condición se deje de cumplir.

$$receivedScreenMessages.Count > frameSkipFactor \times bufferSizes$$

Esto quiere decir que la cola nunca será mayor a $frameSkipFactor \times bufferSizes$ *(frameSkipFactor* viene predefinido con el valor 1.5).

6 IMPLEMENTACIÓN DE LA SOLUCIÓN

En esta sección se describirá como utilizar la aplicación desarrollada. Esta descripción asume que ambos equipos, tanto el cliente como el servidor cuentan con el Software instalado y que cumplen con los requerimientos mínimos puntualizados en 4.2.2.2.

A continuación se procede a explicar cómo utilizar Clairvoyance, partiendo por el cliente y siguiendo con el servidor.

6.1 Cliente

La interfaz desarrollada para el cliente fue pensada para ser utilizada sin la necesidad de un Stylus (el pequeño lápiz que a veces acompaña a estos dispositivos). En general se proveen muy pocos botones para evitar confundir al usuario y se le entregan tips y ayudas en pantalla para asistirlo. En general se evitó sobrecargar los formularios de la aplicación.

La Figura 19 muestra capturas de pantalla del programa cliente funcionando. Estas se encuentran en el orden en que serían desplegadas en una ejecución ideal. A continuación se describe cada una de las pantallas (la letra de la lista numerada corresponde a la letra de la figura):

a) Esta es la pantalla inicial que es mostrada al usuario. Destaca el botón “Buscar Monitores” quien es el que inicia el proceso y establece la conexión o eventual creación de la red MANET.

   El usuario podría abandonar la aplicación haciendo clic en el elemento “Salir” del menú inferior. El menú “Avanzadas” posee características que son de utilidad académica y no debiesen ser considerados si es que la aplicación es implantada a nivel de producción.

b) En esta captura de pantalla se muestra la bitácora del cliente, muy útil para diagnosticar problemas. A esta se puede acceder haciendo clic en la pestaña inferior “Log”.
c) La aplicación salta a esta pantalla cuando el usuario presiona el botón “Buscar Monitores” y sirve para notificarle de que se está conectando a la red MANET.

   Aunque suene ilógico, a mayor número de equipos intentando conectarse, más rápido será el proceso. Esto se debe a que si hay más equipos, será más fácil que dos equipos se encuentren simultáneamente e inicien el proceso de creación de la red.

d) Una vez que se ha conectado a la red, Clairvoyance salta a esta pantalla. En la medida de que los servidores vayan conectándose, estos empezarán a aparecer.

e) Aquí ya ha aparecido un servidor (en este caso WINDBOX). Para establecer el enlace y proyectar la pantalla del Smartphone en el LCD, es necesario hacer doble clic sobre la imagen o el texto.
f) Si es que no existen problemas al establecer el enlace, se saltará a esta pantalla. Aquí se entregan unas cuantas informaciones de utilidad y la posibilidad de terminar el enlace presionando el botón “Terminar Enlace”. Al hacerlo se vuelve a la pantalla d (o e si es que hay monitores disponibles).

   En este punto es posible minimizar Clairvoyance y centrar la atención en aquello que se quiera mostrar a la audiencia. Clairvoyance continuará funcionando en el fondo del sistema al igual que cualquier otra aplicación. Para cerrarlo simplemente vuelva a maximizar esta aplicación y presione el botón “Salir”.

En caso de que a lo largo de la ejecución de la aplicación exista algún problema, esté será notificado con una ventana emergente (Popup Window).

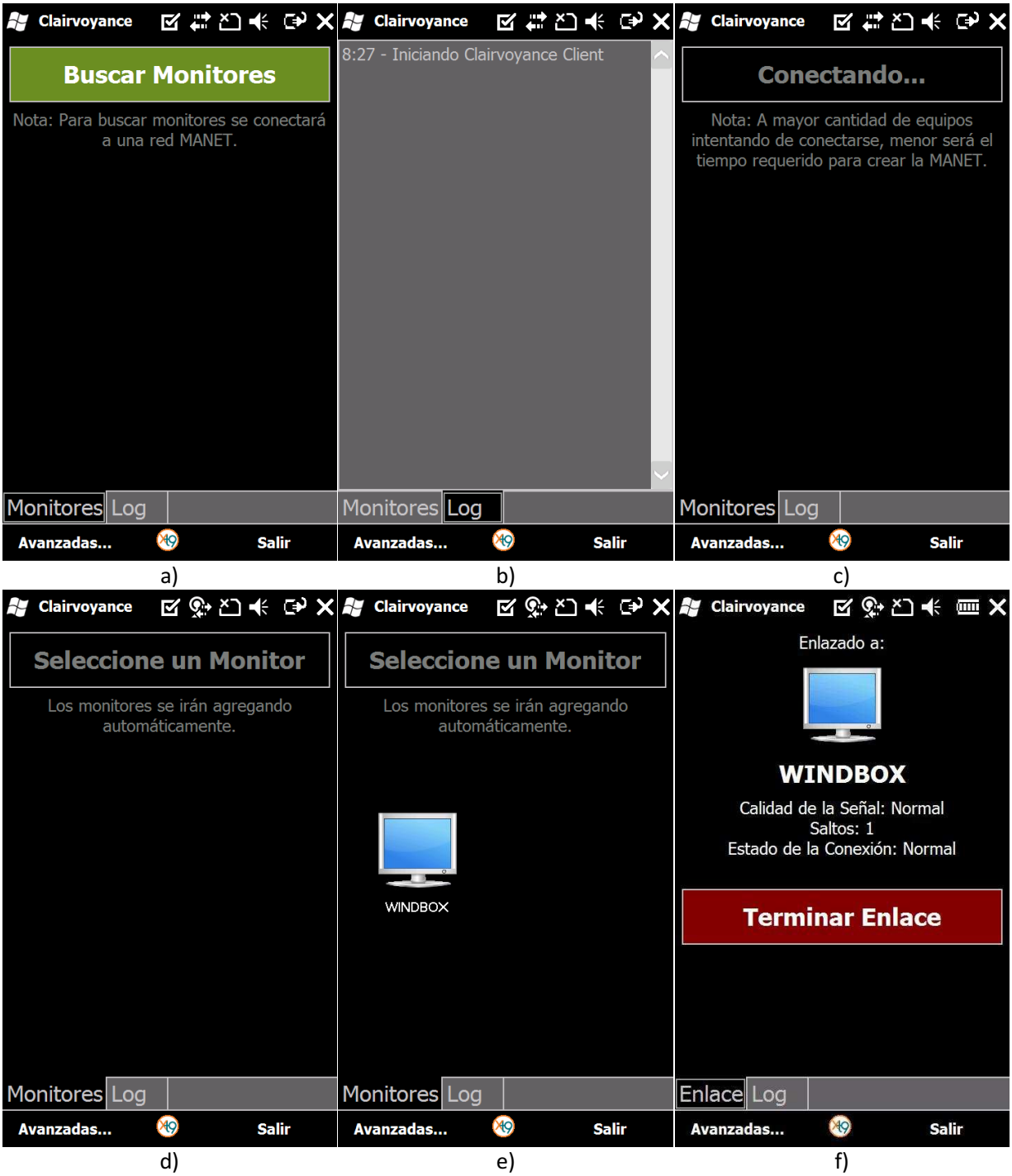
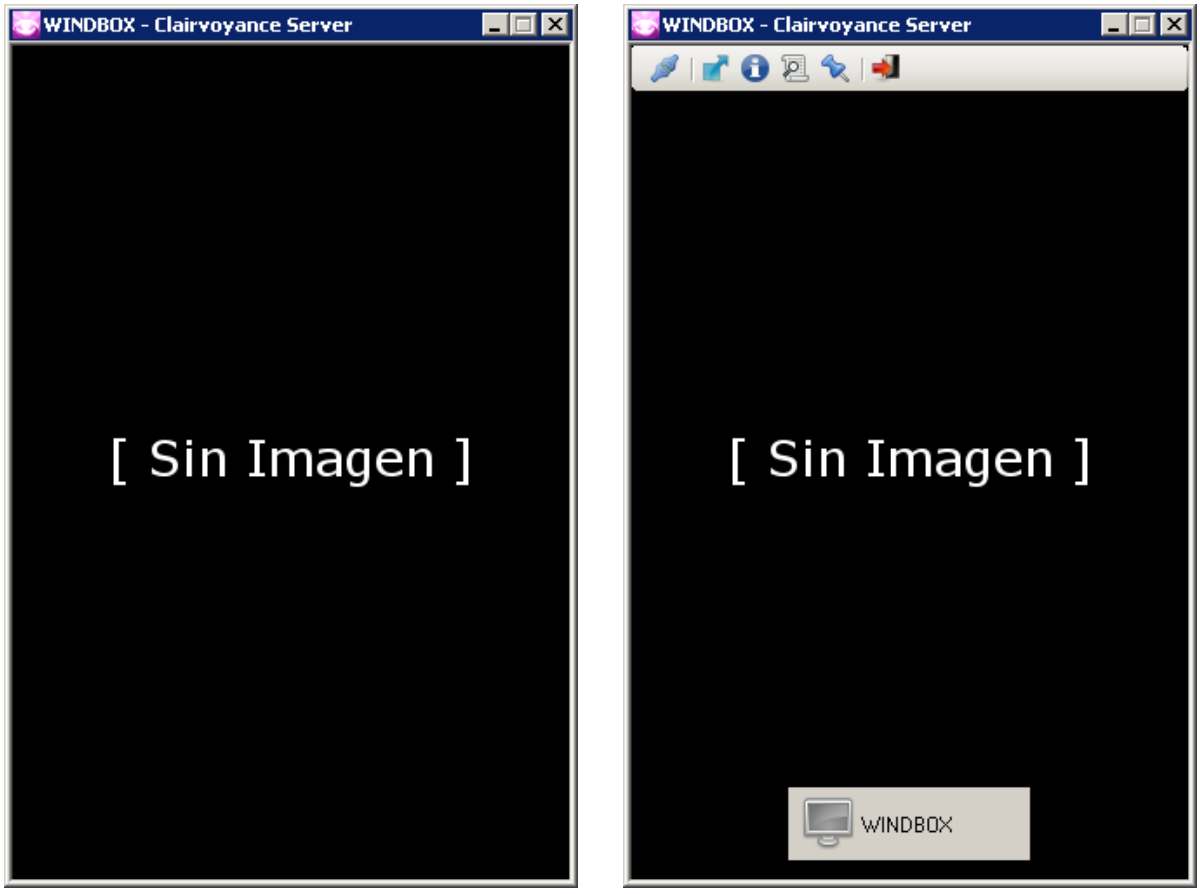


Figura 19. Capturas de pantalla de las diferentes etapas del cliente

6.2 Servidor

El servidor tiene 2 modos de funcionamiento (al menos de la perspectiva de su interfaz gráfica): Modo Ventana y Modo Pantalla Completa. Primero se analizará el Modo Ventana y luego se darán las diferencias que tiene con el Modo Pantalla Completa (que son pocas).

La Figura 20 muestra la aplicación servidor cuando esta se encuentra desconectada, en que destaca el mensaje “Sin Imagen”. La diferencia entre a y b es que en el primero los menús se han desvanecido por inactividad. Al mover el mouse por encima de la aplicación, los menús vuelven a ser mostrados. Se prefirió este enfoque para evitar tener controles innecesarios en la pantalla y es una idea copiada de reproductores de video actuales. Por otro lado, cuando la aplicación está en Modo Pantalla Completa, es práctico que la imagen no se vea contaminada con estos paneles.



a) b)

Figura 20. Desvanecimiento de paneles en modo ventana

La aplicación servidor cuenta con dos paneles: uno superior que tiene botones para realizar acciones y otro inferior, que entrega notificaciones a los usuarios. Los botones que dispone el panel superior son los siguientes (en orden de izquierda a derecha):

1. **Conectar/Desconectar:** Este botón inicia el proceso de conexión y luego deja al servidor esperando peticiones de monitor de los clientes. Es esencialmente el botón que hace funcionar la aplicación. Al presionarse, el botón cambia por otro que permite desconectar al servidor de la MANET.
2. **Pantalla completa/Ventana:** Al presionar este botón, la aplicación cambia de Modo.
3. **Acerca de:** Abre un una pequeña ventana diálogo que entrega informaciones del proyecto.
4. **Log:** Abre la bitácora del servidor Clairvoyance en otra ventana.
5. **Anclar:** Al presionar este botón, los menús no se esconderán. Para que vuelvan a esconderse de forma normal hay que simplemente volver a presionar este botón.
6. **Salir:** Termina la conexión y cierra la aplicación.

El panel de estado (inferior) será descrito siguiendo una ejecución convencional de Clairvoyance. La Figura 21 muestra como este panel va cambiando a lo largo de este ejemplo. A continuación se describe el significado de cada estado, siguiendo el mismo orden que en la figura:

a) El servidor se encuentra desconectado (indicado por el monitor gris). El texto que acompaña al ícono corresponde al nombre del equipo, en este caso WINDBOX.
b) El usuario ha presionado el botón para conectarse a la MANET y el panel cambia de ícono.
c) Al conectarse a la MANET el ícono del Monitor cambia por uno que esta coloreado celeste (emulando que estuviese encendido). En este momento el servidor se encuentra esperando que un cliente se conecte.
d) Un cliente se ha conectado. En este caso es un Smartphone (ver ícono) cuyo nombre es GT-I8000 (El Smartphone Samsung Omnia 2). La calidad de la señal no es muy buena (Estado *Overloaded*), pero esto suele ocurrir inmediatamente después de la conexión. Por otro lado, la ventana del cliente ha cambiado de tamaño para igualar a la resolución del teléfono inteligente.
e) Ahora la conexión se ha normalizado y el indicador de la calidad de señal se muestra verde (Estado *Normal*).
f) A fin de terminar esta demostración mostrando todos los estados, se ha forzado a que la calidad de la señal empeore. Para conseguir esto se ha pasado repentinamente de inactividad a uso intensivo. Producto de esto el indicador de calidad de señal se muestra rojo (Estado *Critical*).

a)

b)

c)

d)

e)

f)

Figura 21. Evolución del panel de estado

El Modo Pantalla Completa es similar, pero guarda dos diferencias notables. La primera es que, al ocupar toda la pantalla, este Modo escala las imágenes para que ocupen la máxima área posible, a diferencia del Modo Ventana en que era la ventana la que tenía que ir cambiando de dimensiones. La segunda diferencia es que las fuentes del panel de notificaciones fueron aumentadas de tamaño, para que la audiencia pueda leer sin mayores inconvenientes. Existen otros cambios pero son de naturaleza

cosmética y no son relevantes. La Figura 22 (con paneles) y la Figura 23 (sin paneles) muestran este Modo.

Un último detalle digno de mención, es que al iniciar la aplicación servidor con el parámetro "/a" este inicia en Modo Pantalla Completa y automáticamente intenta establecer el enlace. Esto es de particular utilidad a fin de dejar esta aplicación funcionando permanentemente en un equipo, sin necesidad de utilizar teclado o mouse para abrirla al iniciarse el Sistema Operativo. La aplicación se agrega al menú de inicio de Windows al ser instalada, con esto el servidor queda configurado para funcionar de forma autómata.

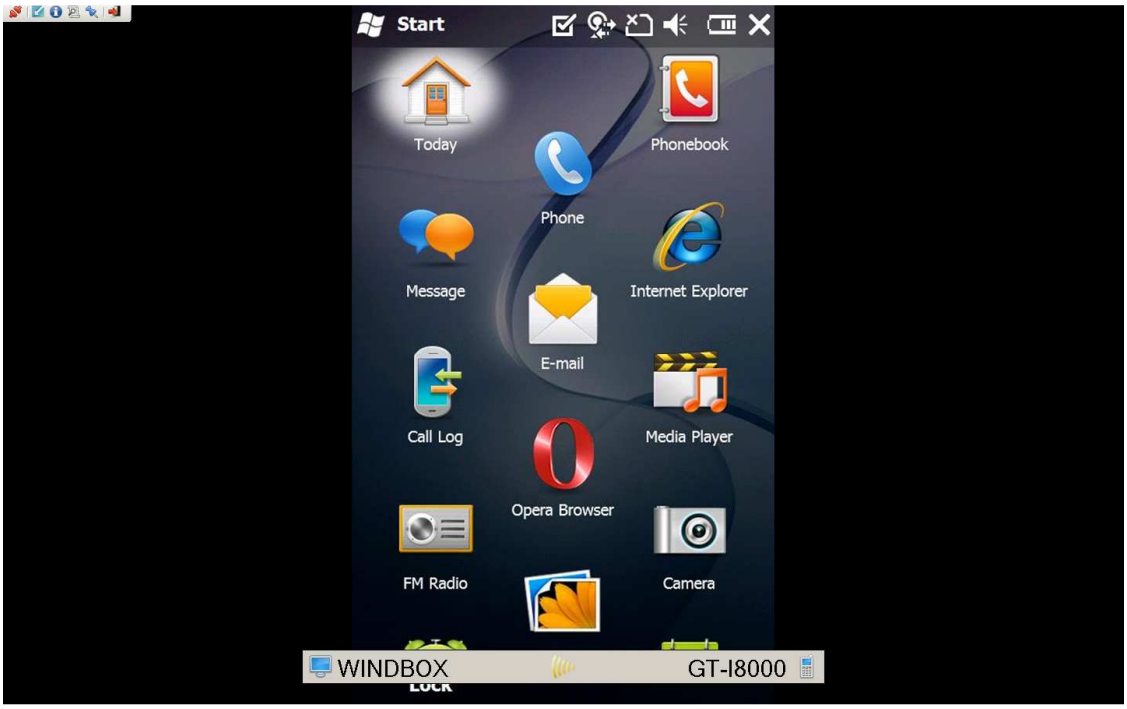


Figura 22. Servidor Clairvoyance en pantalla completa, paneles visibles

Figura 23. Servidor Clairvoyance en pantalla completa, paneles escondidos

7 RESULTADOS OBTENIDOS

A lo largo de esta memoria se ha establecido que ciertos formatos o estrategias para la compresión de imágenes son más rápidos que otras (todo esto en el contexto de la aplicación y el Hardware). En esta sección se fundamentan dichas decisiones.

Lamentablemente, a pesar de tener números concretos, es imposible dictaminar que un formato es sin dudas superior a otro (además es importante la calidad de la imagen, un atributo medianamente subjetivo). Esto ya que no es posible predecir todos los escenarios en que funcionará esta aplicación. Se realiza una serie de suposiciones que, al menos de momento, resultan razonables. En cualquier caso se entregan los resultados, y las herramientas necesarias para volver a calcularlos, a fin de que los distintos formatos puedan ser analizados en una situación futura si es que fuese necesario.

La Tabla 3 muestra el desempeño (en tiempo y espacio) de diferentes formatos de compresión que vienen incluidos en el .Net Compact Framework 3.5. Dichos resultados, solo sirven para tener una idea del orden de magnitud de cada operación.

	tamaño	tipo	tiempo [s]	espacio [kb]	fps	tiempo unitario
BMP	**normal**	**app**	38	1126,4	5,3	0,19
		foto	38	1126,4	5,3	0,19
	pequeño	**app**	5	125,21	40,0	0,025
		foto	5	125,21	40,0	0,025
JPEG	**normal**	**app**	39	36,47	5,1	0,195
		foto	39	35,46	5,1	0,195
	pequeño	**app**	5	7,92	40,0	0,025
		foto	4	4,74	50,0	0,02
PNG	**normal**	**app**	76	83,15	2,6	0,38
		foto	166	241,4	1,2	0,83
	pequeño	**app**	9	12,86	22,2	0,045
		foto	15	32,8	13,3	0,075
GIF	**normal**	**app**	37	27,58	5,4	0,185
		foto	39	71,68	5,1	0,195
	pequeño	**app**	5	5,85	40,0	0,025
		foto	5	12,17	40,0	0,025

Tabla 3. Comparación de los distintos formatos de compresión

En total se realizaron 16 pruebas que consistían en medir el tiempo requerido para almacenar en memoria 200 imágenes. Cuatro pruebas por cada formato (BMP, JPEG, etc.). De estas cuatro, dos son realizadas con imágenes del tamaño de la pantalla del equipo (800x480) y corresponden a las pruebas

tipo “normal”. Las otras dos, tipo “pequeño”, corresponden a pruebas en que se ocupó una imagen nueve veces menor en área (un tercio del alto y un tercio del ancho), de forma similar a como lo hace el Modo Rápido de la estrategia de compresión BDS. Por cada uno de estos tipos (“normal” y “pequeño”) se utilizaron 2 tipos de imágenes; una de un formulario y la otra, una foto. Estas imágenes (ver Figura 24) resultarán familiares al lector, ya que fueron mostradas con anterioridad (por esto se han incluido en dimensiones reducidas acá).

Figura 24. Imágenes ocupadas para las pruebas

El formato BMP fue incluido para tener una idea de lo que costaría (en tiempo de procesamiento y memoria) guardar las imágenes sin ningún tipo de compresión. Los archivos generados tienen un peso excesivo en relación con los otros formatos, pero como es de esperar, son bastante rápidos.

GIF, aunque era una alternativa interesante, tiene un desempeño pobre a la hora de comprimir fotos, rápido, pero utilizando bastante espacio. Por otro lado, la calidad de la imagen es notoriamente peor al resto ya que este formato hace uso de una paleta de colores reducida.

PNG es el formato más lento pero que genera las imágenes de mayor calidad (prácticamente no se notan diferencias con el mapa de bits original). Lamentablemente, lo que se busca no es una calidad excesiva, sino que un compromiso entre calidad y velocidad.

JPEG se alza como el formato por excelencia para esta aplicación por su aceptablemente bajo peso acompañado de tiempos de compresión muy bajos. Resulta impresionante que tome menos tiempo guardar una imagen en el formato JPEG que en BMP (caso pequeño-foto). Esto puede deberse a que JPEG requiere de menos memoria y que en definitiva este sobrecosto pague el tiempo usado para comprimir.

Además de probar los formatos, se estudiaron los costos en tiempo de realizar otras operaciones involucradas con las estrategias de compresión. La Tabla 4 muestra el tiempo requerido para comparar dos mapas de bits 6000 veces. De forma similar a lo anterior, normal se refiere a imágenes del tamaño de la pantalla y pequeño a las versiones reducidas. La subdivisión "resultado" sirve para diferenciar los casos en que se comparaba dos mapas de bits iguales o distintos.

	resultado	tiempo[s]	fps	tiempo unitario
normal	**igual**	55	109	0,0092
	distinta	1	6000	0,00017
pequeño	**igual**	6	1000	0,001
	distinta	0	-	0

Tabla 4. Tiempos de comparación

Si bien las diferencias son grandes, los resultados son esperables. Al ser las imágenes 9 veces menores en término de bits, los tiempos son prácticamente 9 veces menores también. Por otro lado, si las imágenes son distintas, en general es rápido saberlo, al menos para este caso en que las imágenes eran notoriamente distintas.

La Tabla 5 muestra los tiempos que toma a Clairvoyance obtener 2000 capturas de pantalla como mapas de bits.

	tiempo[s]	fps	tiempo unitario
normal	35	57,1	0,0175
escala	19	105,3	0,0095

Tabla 5. Tiempos de captura de pantalla

Con estos datos es posible obtener las limitantes teóricas de la aplicación. Este cálculo es prácticamente de fantasía ya que no considera una serie de factores, entre los cuales destacan:

- Los resultados anteriores se obtuvieron teniendo el procesador y memoria completamente copados. La idea es que el usuario pueda seguir utilizando otras aplicaciones de forma simultánea.
- Existen otras operaciones que no son contabilizadas en este cálculo, como por ejemplo el costo de la red o de otros procesos que se estén ejecutando (otros hilos de ejecución de Clairvoyance por ejemplo).
- Estas pruebas realizadas son sintéticas y sus resultados pueden ser ligeramente distintos a la realidad. Por ejemplo, al realizar las pruebas, el procesador del Smartphone pierde

mucho menos tiempo en cambios de contexto, ya que solo se está realizando una tarea de forma repetitiva.

Con dichas observaciones hechas, se procede a realizar los cálculos. Estos se hacen utilizando los valores de las tablas recientemente descritas. Para obtener el costo de guardar una imagen se utilizó el promedio entre aquella que correspondía a un formulario (app) y a una foto. Básicamente lo que se busca es ver cuánto tiempo toma realizar las operaciones que realiza el método *Tick* de la clase *BDSCompressor*. Este es el ciclo que se ejecuta permanentemente. Fuera del manejo de variables, existen esencialmente 3 casos.

1. Capturar imagen de alta calidad (S_a) y guardarla (G_a).

$$S_a + G_a = 0{,}0175 + 0{,}193 \sim 0{,}22$$

2. Capturar imagen de baja calidad (S_b), compararla con la anterior y obtener que son iguales (C_i).

$$S_b + C_i = 0{,}0095 + 0{,}001 \sim 0{,}011$$

3. Capturar imagen de baja calidad (S_b), compararla con la anterior, obtener que son distintas (C_d) y luego guardar esta imagen de baja calidad (G_b).

$$S_b + C_d + G_b = 0{,}0095 + 0 + 0{,}023 \sim 0{,}0325$$

Finalmente, siendo completamente aventurado, podría decirse que el caso 1 se da un 5% del tiempo, el caso 3 un 20% y el caso 2 se da el resto. Con esto tendríamos que, en promedio, el tiempo que toma realizar un ciclo Tick es:

$$0{,}05 \times 0{,}22 + 0{,}75 \times 0{,}011 + 0{,}2 \times 0{,}0325 \sim 0{,}026$$

Esto quiere decir que, ignorando otras variables, usando el 100% del procesador y en un marco teórico, la estrategia de compresión BDS podría mostrar $^{1}/_{0{,}026} \sim 38{,}4$ frames por segundo en promedio.

Sin embargo, en la práctica y tras varias pruebas, se ha concluido que Clairvoyance funciona bastante bien (cumpliendo los requisitos establecidos en 3.8) con 8 frames por segundo.

8 CONCLUSIONES Y TRABAJO A FUTURO

Con el desarrollo de la aplicación terminado, puede decirse que los objetivos planteados inicialmente, con los requerimientos de software que venían a formalizarlos, fueron cumplidos a cabalidad. Si bien existieron una serie de obstáculos que podrían haber comprometido el resultado final, estos fueron afortunada y rápidamente sorteados al comienzo del proyecto.

De las decisiones de diseño tomadas, es importante recalcar que el uso de la librería HLMP fue una gran ayuda. Si bien este apilamiento de bibliotecas a veces puede llevar a comportamientos extraños, ya que esencialmente son cajas negras que proveen una funcionalidad, en este caso no existieron mayores problemas. El ecosistema y las herramientas de desarrollo de Microsoft estuvieron a la altura de lo esperado, pero sin proveer también mayores ventajas. Windows Phone 7, la siguiente iteración del sistema operativo Windows Mobile (enfocada a Smartphones), será construida desde cero y a priori no sería compatible con aplicaciones desarrolladas para versiones anteriores, lo que imposibilitaría el funcionamiento directo (sin cambios) del trabajo aquí mostrado. Estas son solo especulaciones, ya que aún no se ha lanzado de forma oficial. Lo importante es que la investigación realizada y sus conclusiones trascienden una versión de sistema operativo, por lo que sería erróneo pensar que este trabajo tiene un tiempo de caducidad.

A fin de tener una calidad de video superior, sin sacrificar demasiados recursos, se necesitaría de Hardware especializado. En ese sentido la apuesta de Intel, con su tecnología Wireless Display, se ha asegurado un nivel de Hardware conforme a sus necesidades. Solo resta ver si es que efectivamente arriba al mundo de los Smartphones y en cuanto tiempo.

Lo que es posible aportar, sacar en limpio, tras probar Clairvoyance y evaluar sus fortalezas, debilidades y oportunidades es lo siguiente:

i. Para el éxito de esta tecnología (proyección de pantalla de Smartphone de forma inalámbrica) es necesario que los Smartphones y los LCDs sean realmente ubicuos. Esto no se cumple en la actualidad, por lo que este proyecto queda encapsulado momentáneamente dentro de lo académico.
ii. Resultaría interesante desarrollar esta tecnología para otro tipo de dispositivos como notebook, tablets u otros dispositivos móviles que surjan a futuro. Esto ya que probablemente los Smartphones no serán los únicos equipos con los cuales se trabaje en una sala de reuniones. Por otro lado, probando esta tecnología con estos dispositivos habilita la posibilidad de encontrar nuevas ideas (de momento no evidentes) que encaminen proyectos futuros.
iii. Proyectar la pantalla del Smartphone aunque suficiente, es solo el comienzo. Debiesen investigarse otras formas de aprovechar la pantalla del LCD. Normalmente las pantallas de los Smartphone son más altas que anchas, todo lo contrario que los LCD. Esta

dicotomía causa que el espacio sea pobremente aprovechado. Algunas ideas para mejorar esto:

a. Al girar el teléfono se obtiene una relación de aspecto (ancho y alto) similar a la de un LCD. El inconveniente de esto es que el cambio podría no verse reflejado en la información gráfica que se guarda de la pantalla (ver Figura 4). Podría incluirse en Clairvoyance una componente que esté atenta a este cambio de orientación del teléfono y que corrija la imagen.
b. Podría darse una solución "manual" al usuario, en que este pueda rotar las imágenes proyectadas en 90 grados a la izquierda o derecha presionando botones en la aplicación cliente.
c. Una solución simple sería, de forma similar a lo descrito en b, poner botones que permitan girar la pantalla en el servidor y a su vez girar también el LCD (el aparato físico). De este modo se tendría una pantalla más grande que tiene una relación de aspecto similar.

Otra alternativa más radical para utilizar el espacio en pantalla podría ser permitir a varios Smartphones conectarse simultáneamente a un servidor.

iv. Crear una aplicación que capture la pantalla y que la proyecte en un LCD es una solución simple pero incompleta al problema. La solución óptima sería que el Sistema Operativo provea esta funcionalidad de forma transparente. De este modo las aplicaciones deberían considerar la eventualidad de que la pantalla sea proyectada y reaccionar al respecto (de forma similar a como algunas aplicaciones reaccionan cuando se gira el teléfono). Si bien cada aplicación podría hacer esto de forma independiente, sería un enfoque desordenado (y con mucha redundancia), por lo que es preferible integrar esto dentro del Sistema Operativo. Por ejemplo, una aplicación para leer libros podría escuchar eventos de este tipo (iniciando-proyección y terminando-proyección con sus respectivos parámetros) y cuando sea proyectada en un LCD no solo aumentar en tamaño de forma lineal sino que además podría mostrar más líneas, como si efectivamente el tamaño de la pantalla hubiese cambiado. La pantalla del Smartphone, en lugar de mostrar la aplicación, podría dedicarse únicamente a tener controles para desplazarse dentro del libro. La Figura 25 ilustra este ejemplo de una aplicación que está consciente de que está siendo proyectada y reacciona de forma acorde. En la figura, a muestra cuando la aplicación solo está en el Smartphone y b cuando es proyectada.

Con esta medida la idea es entregar a las aplicaciones la libertad y responsabilidad de reaccionar dependiendo de su función particular.

Para conseguir esto se necesita por un lado de un trabajo importante en el Sistema Operativo a bajo nivel y por otro lado de que las aplicaciones incluidas (e idealmente todas) consideren la eventualidad de que sean proyectadas. El Sistema Operativo podría

proveer soluciones estándar para los casos más comunes. Por ejemplo la pantalla podría expandirse (como lo hace Clairvoyance) o bien la aplicación podría crecer a una resolución mayor (de forma similar a cuando es maximizada en un computador de escritorio).

En un equipo que provea esta funcionalidad probablemente harán falta ciertos tipos de aplicaciones nuevas, como por ejemplo una aplicación puntero para poder dirigir la atención de la audiencia desde el teléfono o bien una aplicación lupa.

a) b)

Figura 25. Ejemplo de aplicación que está consciente de proyección

Las predicciones establecen que los Smartphones nos acompañarán a todos lados y que serán nuestra interfaz de comunicación con un omnipresente mundo tecnológico. Más allá de los objetivos que fueron planteados, y en términos más abstractos, la idea de fondo de esta memoria es proveer una herramienta que apoye a la sala de reuniones del futuro. Clairvoyance es un proyecto constituido en dicha línea. El autor de esta memoria tiene la convicción de que, aunque este proyecto puntual muy probablemente no alcance la masa crítica de usuarios como para volverlo popular, es importante estar un paso al frente. La Ingeniería en Ciencias de la Computación demanda estar en primera línea en el mundo tecnológico, observar las tendencias e investigarlas, en este enfoque se tiene la satisfacción de que esta memoria fue una valiosa y fructífera experiencia de investigación y desarrollo.

BIBLIOGRAFÍA

1. **comScore**. [En línea] 5 de Abril de 2010. [Citado el: 23 de Junio de 2010.] <http://www.comscore.com/Press_Events/Press_Releases/2010/4/comScore_Reports_February_2010_U.S._Mobile_Subscriber_Market_Share>.

2. **Scanlon, J.** If walls could talk, streets might join in. *New York Times.* September 18, 2003.

3. **Churchill, E. Nelson, L. and Denoue, L.** *Multimedia Fliers: Information Sharing With Digital Community Bulletin Boards.* Palo Alto, CA, USA. Wolters Kluwer Academic Publishers, pp. 97-117, September 2003.

4. **Greenberg, S., Boyle, M. and LaBerge, J.** *PDAs and Shared Public Displays: Making Personal Information Public, and Public Information Personal.* Personal Technologies. Springer-Verlag London Ltd, 1999.

5. **McCarthy, J.** *Using public displays to create conversation opportunities.* Workshop on Public, Community, and Situated Displays at the ACM 2002 Conference on Computer Supported Cooperative Work (CSCW 2002). New Orleans 2002.

6. **Russell, D.M., Drews, C. and Sue, A.** *Social Aspects of Using Large Public Interactive Displays for Collaboration.* Springer-Verlag, LNCS Vol. 2498, pp. 229-236, 2002.

7. **Seokhee Jeon, Jane Hwang, Gerard J. Kim, Mark Billinghurst.** *Interaction Techniques in Large Display Environments using Hand-held Devices.* VRST'06. Limassol, Cyprus, 2006.

8. **Myers, Brad A.** *Using Handhelds and PCs Together.* Communications of the ACM Vol. 44, pp. 34-41, November 2001.

9. **Paek, T, M. Agrawala, S. Basu, S. Drucker, T. Kristjansson, R. Logan, K. Toyama, A. Wilson.** *Toward Universal Mobile Interaction for Shared Displays.* Proceedings of CSCW. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW'04). pp. 266-269. Chicago, Illinois, USA, November 6–10, 2004.

10. **Wikipedia**. [En línea] 31 de Mayo de 2010. [Citado el: 21 de Junio de 2010.] <http://es.wikipedia.org/wiki/Clarividencia>.

11. **J., Rodríguez.** *Infraestructura de Trabajo Colaborativo Móvil para Inspección Técnica de Obras.* Departamento de Ciencias de la Computación, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas. Tesis de Magister. Santiago, Chile, 2010.

12. **Rodríguez-Covili, J., Ochoa, S., Pino, J.** *Enhancing Mobile Collaboration with HLMP.* Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD'10). IEEE Press, Shanghai, China, April 2010.

13. **Rodríguez-Covili, J., Ochoa, S., Pino, J., Messeguer, R., Medina, E., Royo, D.** *HLMP API: A Software Library to Support the Development of Mobile Collaborative Applications.* Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD'10). IEEE Press, Shanghai, China, April 2010.

ANEXO 1: CÓDIGO FUENTE DE LA SOLUCIÓN

A continuación se incluye el código fuente de la clase *CommunicationController.cs* del cliente.

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using CommLayerCompact;
using Clairvoyance.Client.Net;
using Clairvoyance.Client.Screencast;

namespace Clairvoyance.Client
{
    /// <summary>
    /// Clase encargada de orquestar toda la comunicación del cliente
    /// </summary>
    class CommunicationController
    {
        /// <summary>
        /// Clase que encapsula la configuración de la MANET.
        /// </summary>
        private Configuration configuration;

        /// <summary>
        /// Clase encargada de administrar la comunicación de la MANET.
        /// </summary>
        private static Communication communication;

        public static Communication Communication
        {
            get { return communication; }
            set { communication = value; }
        }

        /// <summary>
        /// Protocolo del cliente.
        /// </summary>
        private Protocol protocol;

        /// <summary>
        /// Formulario principal de la aplicación cliente.
        /// </summary>
        private MainForm mainForm;

        /// <summary>
        /// Clase auxiliar encargada de establecer los enlaces con el servidor.
        /// </summary>
        private Linker linker;

        /// <summary>
        /// Clase auxiliar encargada de emitir las pantallas del cliente al servidor.
        /// </summary>
        private ScreenEmitter emitter;
```

```
        /// <summary>
        /// Inicializa todas las variables de comunicación.
        /// </summary>
        /// <param name="mainForm"></param>
        public CommunicationController(MainForm mainForm)
        {
            this.mainForm = mainForm;
            mainForm.WriteLog("Iniciando Clairvoyance Client");

            try
            {
                configuration = new Configuration();

                // La siguiente variable debe ser definida acrode al equipo objetivo
                configuration.NetData.OpSystem =
NetLayerCompact.OpSystemType.SAMSUNGOMNIAII;

                UserDataAdmin.Init(configuration.NetUser, false);

                SubProtocolList subProtocols = new SubProtocolList();

                // Configuración de protocolo y auxiliares
                protocol = new Protocol();
                protocol.Log += mainForm.WriteLog;
                protocol.ScreenSendingError += this.ScreenSendingErrorHandler;
                protocol.ScreenResponseReceived += this.ScreenResponseHandler;

                // Construyo las clases auxiliares
                linker = new Linker(protocol);

                subProtocols.add(Net.Constants.ProtocolType, protocol);

                communication = new Communication(configuration, subProtocols, null);

                //Eventos de conexión a la red
                communication.connectingEvent += ConnectingHandler;
                communication.connectEvent += ConnectHandler;
                communication.reconnectingEvent += ReconnectingHandler;

                //Eventos relacionados con los usuarios de la manet
                communication.removeUserEvent += RemoveUserHandle;
                communication.addUserEvent += CheckUser;
                communication.refreshUserEvent += CheckUser;
                communication.refreshUserEvent += UpdateLinkPage;

                //Eventos relacionados con el comportamiento de la librería
                communication.exceptionEvent += ExceptionHandler;
                communication.netInformationEvent += mainForm.WriteLog;

                communication.startEventConsumer();
            }
            catch (Exception e)
            {
                mainForm.ShowMessageBox(e.Message, "Error al Iniciar");
                mainForm.WriteLog(e.Message);
                Close();
            }
        }
```

```
        /// <summary>
        /// Termina abruptamente el enlace y reestablece los valores iniciales.
variables de protocolo y emisor (en caso de que hubiese).
        /// </summary>
        private void KillLink()
        {
            protocol.KillLink();
            if (emitter != null)
            {
                emitter.Close();
                emitter = null;
            }
        }

        /// <summary>
        /// Termina el enlace notificando al servidor y cambiando la UI.
        /// </summary>
        public void CloseLink()
        {
            protocol.SendUnlinkMessage();
            KillLink();
            mainForm.ChangeUIState(MainForm.UIState.DisplaySelection);
        }

        /// <summary>
        /// Termina la sesion y destruye todo.
        /// </summary>
        public void Close()
        {
            Disconnect();
            if (communication != null)
                communication.stopEventConsumer();
        }

        /// <summary>
        /// Termina la sesion.
        /// </summary>
        public void Disconnect()
        {
            CloseLink();
            if (protocol != null)
                protocol.CloseLink();
            if (communication != null)
            {
                communication.disconnect();
            }
            mainForm.ChangeUIState(MainForm.UIState.Disconected);
        }
```

```
        /// <summary>
        /// Inicia la conección a la MANET
        /// </summary>
        public void Connect()
        {
            try
            {
                communication.connect();
            }
            catch (Exception e)
            {
                mainForm.WriteLog(e.StackTrace);
                mainForm.ShowMessageBox(e.Message, "Error al Conectar a MANET");
            }
        }

        #region Connection Handles

        /// <summary>
        /// Invocado cuando el usuario se quiere conectar a la MANET (manual)
        /// </summary>
        private void ConnectingHandler()
        {
            mainForm.WriteLog("Conectando a MANET...");
            mainForm.ChangeUIState(MainForm.UIState.Connecting);
        }

        /// <summary>
        /// Invocado cuando el cliente se conecta
        /// </summary>
        private void ConnectHandler()
        {
            mainForm.WriteLog("Conectado a MANET");
            mainForm.ChangeUIState(MainForm.UIState.DisplaySelection);
        }

        /// <summary>
        /// Invocado cuando el usuario se intenta reconectar a la MANET (automático)
        /// </summary>
        private void ReconnectingHandler()
        {
            mainForm.WriteLog("Reconectando a MANET...");
            KillLink();
            mainForm.ChangeUIState(MainForm.UIState.Connecting);
        }

        /// <summary>
        /// Invocado cuando HLMP se encuentra en una situación de excepción
        /// </summary>
        /// <param name="e">Excepción arrojada</param>
        private void ExceptionHandler(Exception e)
        {
            mainForm.ShowMessageBox(e.Message + System.Environment.NewLine + "Cerrando
enlace", "Error en la comunicación");
            mainForm.WriteLog(e.StackTrace);
            KillLink();
            mainForm.Close();
        }

        #endregion
```

```
        #region Screen Emittion

        /// <summary>
        /// Invocado por HLMP cuando un usuario ha cambiado su estado. Se encarga de
actualizar el Tab link de la aplicación, con información del estado de la red.
        /// </summary>
        /// <param name="user"></param>
        private void UpdateLinkPage(NetUser user)
        {
            if (protocol.CurrentStatus == Protocol.Status.Linked && user.Id ==
protocol.LinkedUser.Id)
            {
                mainForm.UpdateLinkPage(user.Name, user.SignalQuality, user.JumpsAway,
communication.Configuration.NetUser.State);
            }
        }

        /// <summary>
        /// Invocado cuando hay un error al enviar un mensaje tipo Screen.
        /// </summary>
        private void ScreenSendingErrorHandler()
        {
            KillLink();
            mainForm.WriteLog("Error al comunicarse con servidor. Terminado Enlace.");
            mainForm.ShowMessageBox("Error al comunicarse con servidor. Terminado
Enlace.", "Error en emisión");
            mainForm.ChangeUIState(MainForm.UIState.DisplaySelection);
        }

        /// <summary>
        /// Invocado cuando el servidor recibe un mensaje tipo Screen. Se utiliza para
terminar el enlance del lado del cliente cuando el servidor rechaza estos mensajes.
        /// </summary>
        /// <param name="netUser">Usuario Servidor</param>
        /// <param name="answer">Valor booleano con la respuesta</param>
        /// <param name="reason">Razon del Servidor</param>
        private void ScreenResponseHandler(NetUser netUser, bool answer, String reason)
        {
            if (!answer)
            {
                KillLink();
                mainForm.WriteLog("Servidor rechazo conección. Terminado Enlace.");
                mainForm.ShowMessageBox("Servidor rechazo conección. Terminado
Enlace.", "Error en emisión");
                mainForm.ChangeUIState(MainForm.UIState.DisplaySelection);
            }
        }

        #endregion
```

```
        #region Link

        /// <summary>
        /// Establece el enlace con un Servidor
        /// </summary>
        /// <param name="guid">Identificador del Servidor con el cual establecer el
enlace</param>
        /// <param name="resolution">Resolución de pantalla del dispositivo
cliente</param>
        public void Link(Guid guid, System.Drawing.Rectangle resolution)
        {
            NetUser[] users = communication.NetUserList.userListToArray();
            NetUser selectedUser = null;
            foreach (NetUser user in users)
                if (user.Id == guid)
                    selectedUser = user;
            if (selectedUser == null)
            {
                mainForm.WriteLog("Usuario no encontrado en lista de usuarios de
Communication");
                mainForm.ShowMessageBox("Usuario no encontrado en lista de usuarios de
Communication", "Error al Enlazar");
                mainForm.RemoveDisplay(guid);
            }
            CompressionStrategies compressionStrategies =
CompressionStrategies.Instance;
            if (linker.Link(selectedUser, resolution, compressionStrategies.Current))
            {
                mainForm.UpdateLinkPage(selectedUser.Name, selectedUser.SignalQuality,
selectedUser.JumpsAway, communication.Configuration.NetUser.State);
                mainForm.WriteLog("Enlazado a " + selectedUser.Name);

                mainForm.ChangeUIState(MainForm.UIState.Linked);
                emitter = new ScreenEmitter(protocol, mainForm.WriteLog);
            }
            else
            {
                mainForm.WriteLog("Error al establecer enlace con: " +
selectedUser.Name);
                mainForm.ShowMessageBox("Error al establecer enlace con: " +
selectedUser.Name, "Error al Enlazar");
                mainForm.RemoveDisplay(guid);
            }
        }

        #endregion
```

```
        #region Display Search

        /// <summary>
        /// Invocado cuando un usuario abandona la MANET. Se utiliza para remover
servidores de la lista.
        /// </summary>
        /// <param name="user">Usuario a remover</param>
        private void RemoveUserHandle(NetUser user)
        {
            mainForm.WriteLog("RemoveUserHandle: "+user.Name);
            mainForm.RemoveDisplay(user.Id);
        }

        /// <summary>
        /// Invocado cuando un usuario entra o cambia de estado en la MANET. Revisa si
un usuario es un servidor disponible y en dicho caso lo agrega a la lista.
        /// </summary>
        /// <param name="user">Usuario a revisar</param>
        private void CheckUser(NetUser user)
        {
            if (UserDataAdmin.IsServer(user))
            {
                if (UserDataAdmin.IsAvailable(user))
                    mainForm.AddDisplay(user.Id, user.Name);
                else
                    mainForm.RemoveDisplay(user.Id);
            }
        }

        #endregion
    }
}
```