Marte 3.0: Una videoconferencia 2.0

J. Cerviño, P. Rodríguez, J. Salvachúa, G. Huecas y F. Escribano

Resumen— Este artículo describe el diseño e implementación de un sistema de colaboración multimedia basado en la utilización de clientes ligeros desplegados en la *web*. Su objetivo es definir una nueva arquitectura de conferencias en *Internet* centrada en la facilidad de uso y de instalación, dotando de conectividad total al usuario final con el resto de participantes. A lo largo del documento presentaremos las distintas soluciones de partida, discutiremos las decisiones de diseño citando las ventajas del nuevo modelo y mostraremos aquellos problemas encontrados en la implementación que son inherentes a este tipo de escenarios.

Palabras clave— Trabajo cooperativo asistido por ordenador (*Computer Supported Cooperative Work*), aplicaciones ricas de Internet (*Rich Internet Applications*), comunicación multimedia (*Multimedia communication*), sistemas cliente-servidor (*Client-server systems*), videoconferencia (*Videoconferencing*).

I. Introducción

HASTA ahora las aplicaciones multimedia distribuidas en la red de redes (*Internet*) son de una complejidad de desarrollo muy importante, a la dificultad intrínseca de cualquier aplicación distribuida en *Internet*, hay que sumar los requisitos de retardo propios de las aplicaciones multimedia en tiempo real donde el tiempo de respuesta es muy importante de cara a obtener un resultado satisfactorio. Por todo lo anterior, este tipo de aplicaciones generalmente quedaban reducidas a un entorno bastante restringido con gran ancho de banda y capacidad de proceso del servidor. Sin embargo, ante el constante aumento del ancho de banda disponible y de capacidad de proceso de las máquinas, estas utilidades han ido ampliando su interés de cara a un público más general lo que, a su vez, ha fomentado el desarrollo de nuevas plataformas para su desarrollo que disminuyen su complejidad y aumentan la facilidad de instalación y uso.

En el siguiente apartado comentaremos las experiencias previas dentro del mismo grupo, con esto en el apartado III se pretende definir la aplicación de videoconferencia diseñada comentando los diferentes problemas encontrados. En el apartado IV presentaremos las conclusiones del trabajo realizado y propondremos una serie de proyectos futuros.

G. Huecas y J. Salvachúa imparten docencia en el Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid, c/ Ciudad Universitaria s/n, E.T.S.I. Telecomunicación, 28049, Madrid.

J. Cerviño y F. Escribano realizan sus estudios de doctorado en el mismo centro.

J. Cerviño, P. Rodríguez y F. Escribano son becarios de investigación asociados al mismo departamento, (correos e.: ghuecas@dit.upm.es; jsr@dit.upm.es, jcervino@dit.upm.es, prodriguez@dit.upm.es, fec@dit.upm.es).

II. Problemática y experiencias anteriores

A. Marte 1.0

La primera versión de *Marte* se desarrolló en el año 2004 como una alternativa a los servicios de colaboración multimedia que existían en aquel momento. Los servicios que se encontraban entonces tenían requisitos muy fuertes en cuanto a los sistemas que tenían que soportarlos, de forma que era necesario disponer de máquinas enteras y acceso directo a *Internet* para poder crear conferencias multimedia entre varios participantes.

Todos los objetivos se encuadraron en intentar crear un servicio orientado al usuario medio de *Internet*, de forma que se aumentara la facilidad de uso (con interfaces de usuario sencillos, aplicaciones clientes no muy pesadas), que lograra la comunicación teniendo en cuenta el bajo ancho de banda y dispositivos tales como los traductores de direcciones de red (NAT) y cortafuegos (*Firewalls*), y que fuera un sistema fácilmente escalable utilizando soluciones estandarizadas.

El resultado fue una arquitectura centralizada, mostrada en la Fig. 1, basada en el protocolo de inicio de sesión (SIP) y el protocolo de transporte de tiempo real (RTP) en la que se utilizaron dispositivos de retransmisión (*proxies*) SIP para la señalización y con los que los clientes podían evitar la problemática de los dispositivos NAT. Se escogió SIP frente a otros protocolos H.323 debido al auge que en aquellos momentos experimentaba la tecnología, ya que era (y continúa siendo) el mayor valuarte de la convergencia entre *Internet* y las redes móviles. El servidor se ejecutaba en máquinas *Linux* y se desarrolló en el lenguaje C para conseguir mayor eficiencia en tiempo de ejecución del código. En la parte cliente se tenía un ejecutable instalable en el sistema operativo *Microsoft* Windows y desarrollado bajo la plataforma .NET que se comunicaba con el servidor para la señalización y que permitía establecer sesiones de mensajería instantánea, vídeo, audio y compartición de escritorio con los demás usuarios conectados. El diseño del código se intentó dividir en módulos eficazmente con el objetivo de poder extender más adelante su funcionalidad.

Por tanto la solución pasó por crear una doble pila en la que por una parte estaba la señalización de la comunicación (en la que como hemos comentado se utilizó SIP) que se basaba en una arquitectura centralizada y por otra el envío del flujo de datos utilizando RTP a través de las unidades de control multipunto (MCU), que se encargaba de unir diferentes flujos de vídeo y audio de diferentes conexiones.

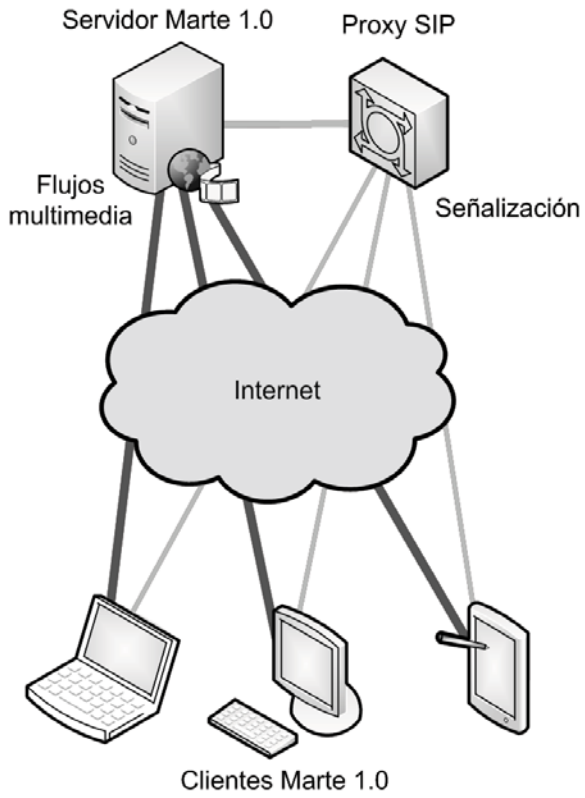

Fig. 1 Arquitectura de Marte 1.0, basada en SIP y RTP

Entre los inconvenientes que presentaba este sistema encontrábamos que no era un sistema de conferencias multimedia en el que los usuarios podían utilizar diferentes salas de conferencias multimedia, y que los sistemas que el *Internet Engineering Task Force* (IETF) había definido por entonces para compatibilizar SIP con los dispositivos NAT no resolvían los problemas en todos los entornos por lo que la comunicación no era siempre posible.

Como característica importante del cliente de cara al interfaz de usuario, se ofrecían diferentes modos de interacción que aseguraban a un cliente que el resto de los participantes en la conferencia veían exactamente lo mismo en sus pantallas en un momento determinado.

B. Marte 2.0

La siguiente versión de *Marte* trató de solventar los problemas comentados en el punto anterior utilizando la misma arquitectura. De esta forma se mejoró la respuesta del sistema frente a la existencia de *Firewalls* utilizando túneles que llevaban el tráfico entre los clientes, se añadió la capacidad para servir múltiples conferencias simultáneamente y se añadió un sistema de presencia más avanzado. Además de la presencia también se añadió un sistema de autenticación y gestión de usuarios basada en el protocolo de acceso a directorios ligeros (LDAP).

En la parte del cliente también se modificó la interfaz del usuario para hacerla aún más sencilla de cara al usuario final, de forma que toda la aplicación hacía uso de una única ventana, en la que se iban añadiendo las ventanas de mensajería, vídeo y audio y compartición de escritorio. Y se introdujo el servicio de pizarra compartida que podían usar los usuarios conectados a una sala de conferencia, servicio visto en productos conocidos como *Yahoo*! *Messenger* o *Microsoft MSN Messenger*. Los modos de interacción existentes se ampliaron para dar cabida a las nuevas capacidades de la aplicación.

Se incluyó un sistema de control de conferencias propietario con el que se podían gestionar las diferentes salas de conferencia multimedia, la arquitectura general quedó como indica la Fig. 2.

Pero el hecho de incorporar elementos complejos en el cliente hace que la aplicación que deben instalar los usuarios finales sea cada vez más compleja, por lo que se pierde facilidad de uso. Además el hecho de que las conexiones de líneas de subscripción digital (xDSL) cada vez tengan mayor ancho de banda hace que pierda fuerza el requisito de ahorrar ancho de banda para poder obtener mayor calidad en el servicio.

En resumen, el problema al que nos enfrentamos en las versiones anteriores de Marte no era el ahorro de ancho de banda mediante el uso de complejas codificaciones y protocolos de nivel de transporte, si no que al contrario sacrificaremos si es necesario este ahorro e incluso la calidad de los datos multimedia en favor de la conectividad. La idea que surge a partir de *Marte* 2.0 es conseguir un sistema cliente/servidor en el que la mayor carga de trabajo resida en el servidor mientras que los clientes sean muy ligeros en cuanto a computación y, por otra parte, se intenta lograr éxito en la conectividad entre todos los clientes en el mayor número de escenarios posible, por lo que nos centraremos en comunicaciones TCP en vez de UDP (por lo que como veremos tendremos que abandonar la implementación de SIP). Para conseguir esta conectividad nos apoyamos como explicaremos en el siguiente apartado en el buque insignia de la conectividad que hoy en día arrasa en todo *Internet*: la *web* 2.0.

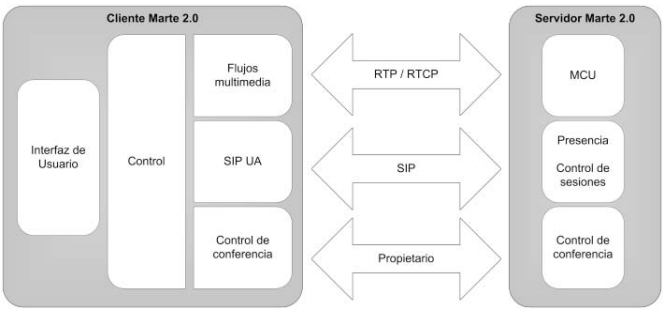

Fig. 2 Arquitectura de Marte 2.0, con servicio de presencia

III. Arquitectura de Marte 3.0

A. Arquitectura General

A la vista de los problemas encontrados en las versiones anteriores de *Marte*, se comenzó a diseñar una nueva arquitectura más sencilla que basada en los puntos que iremos explicando en este apartado.

1) Entorno para web 2.0

El primer objetivo fue encontrar un entorno de desarrollo que permitiera crear esta arquitectura de conferencias multimedia de

forma que fuera capaz de ejecutarse en el mayor número de escenarios posible. Por eso centramos la base de nuestro estudio no en el servidor, ni en la tecnología del servidor, sino en los distintos clientes y en las posibilidades que ofrecía cada uno de cara a videoconferencias. Para ello la solución más interesante que encontramos fue la de crear un cliente accesible vía *web* y que pudiera ser ejecutado en el mayor número de casos posible, incluyendo la adptación sencilla de dispositivos móviles. Con esta premisa se presentaron dos tecnologías posibles: los subprogramas interactivos (*applets*) Java de *Sun* y *Adobe Flash*.

La tecnología de Java se descartó rápidamente debido a que en experiencias previas no se obtuvo buen resultado con la biblioteca preparada para codificar vídeo y audio (*Java Media Framework* [10]), debido a su alto consumo de recursos y lenta respuesta. Además este entorno lleva mucho tiempo sin ser actualizado [1] con las últimas codificaciones por lo que se ha quedado obsoleto. Sin embargo uno de sus puntos fuertes es el carácter abierto que presenta y las distintas posibilidades que ofrece de cara a utilizar distintas tecnologías de servidor. Existen hoy en día múltiples de proyectos maduros que se basan en tecnología Java para conseguir comunicación multimedia entre distintos clientes utilizando arquitecturas tanto centralizadas como distribuidas.

Por otro lado la tecnología *Flash* de *Adobe* [11] presenta un entorno de carácter abierto desde hace ya varios años que permite crear aplicaciones *web* de forma sencilla, pero con capacidades multimedia muy potentes. Este entorno se llama *Adobe Flex*, y se basa en la utilización de dos lenguajes de desarrollo diferentes: un lenguaje de marcado multimedia extensible (MXML [9]) para el diseño de interfaces de usuario y *ActionScript* para el desarrollo de la lógica de aplicación. Para los desarrolladores presenta un API con gran cantidad de opciones multimedia muy potentes, que permite de forma sencilla y en pocos pasos crear conferencias de audio y vídeo entre varios participantes; eso sí, utilizando siempre servidores multimedia, ya sean de Adobe o de terceras partes. Por lo que el inconveniente más importante es que obliga a la utilización de una arquitectura centralizada en la que los flujos multimedia siempre deben viajar por el servidor.

2) Escritorio compartido

El segundo objetivo fue enriquecer la arquitectura lo máximo posible añadiendo servicios al audio y el vídeo. Para ello entre los flujos también deberíamos incluir el de compartición de escritorio, de forma que fuera posible crear salas de conferencia en las que uno o varios usuarios pudieran mostrar las ventanas que había en el escritorio. Este servicio es muy útil en conferencias en las que existe un ponente que quiere hacer presentaciones ayudándose de transparencias o diapositivas. La tecnología que se escogió para este propósito fue la computación de red virtual (VNC [7]), que es la que se utilizó con éxito en las versiones anteriores debido a que es una tecnología muy madura que da muchas posibilidades. Sin embargo esta tecnología de por sí propone conexión punto a punto, sin utilizar servidores o *proxies*. Como en anteriores ocasiones nosotros utilizamos un *proxy* intermedio que nos permite ahorrar ancho de banda en los clientes (al no enviar el servidor VNC la captura de pantalla a cada uno de los clientes VNC) y lograr éxito en la mayor número de escenarios de conexión.

3) Arquitectura de servidor centralizada

Como dijimos anteriormente el uso de la tecnología *Flash* obliga actualmente a implementar una arquitectura centralizada en un servidor multimedia. Por defecto este servidor sería el producto de pago de *Adobe* denominado *Adobe Flash Media Server*. Este servidor permite desarrollar aplicaciones servidoras programando con *ActionScript* servicios de comunicación en tiempo real entre clientes. Además ofrece muchas más características, pero que no son necesarias para los objetivos de nuestro sistema. El problema de este producto es su carácter privado y cerrado ya que no permite interactuar con aplicaciones de terceros, por lo que con su utilización se complicarían otros objetivos como el de escritorio compartido (explicado anteriormente) o el de un sistema abierto (que explicaremos más adelante). Sin embargo existe otra alternativa muy seria a la utilización de este producto, que es *Red5*, un servidor de código abierto que presenta casi todas las posibilidades que *Flash Media Server*, pero con la ventaja de utilizar Java para el desarrollo de la lógica de aplicación y el de ser completamente abierto. En la arquitectura de servidor veremos las ventajas e inconvenientes de esta solución con más detalle y los protocolos de los que hacen uso los dos productos, pero adelantaremos aquí que hacen uso de un protocolo cliente/servidor en el que tanto los mensajes como los flujos multimedia van por la misma conexión, este protocolo (creado por *Adobe*) se denomina protocolo de mensajería en tiempo real (RTMP) [12].

4) Interfaz de usuario sencilla

La idea de utilizar un cliente *web* permite acercar este servicio al usuario medio de *Internet*, el cual podría tener conocimientos muy escasos sobre las aplicaciones de videoconferencia. Por lo que un requisito importante fue el de crear una interfaz sencilla e intuitiva al usuario, basada en el hecho de que crear, eliminar y unirse a conferencias sea una tarea rápida y directa y el hecho de hablar con los demás usuarios de cada sala de conferencia no requiera conocimientos multimedia avanzados.

5) Sistema abierto

El último objetivo es herencia de la primera versión de *Marte*, que se basaba en la creación de un sistema flexible y escalable. Para ello la mejor manera es crear una aplicación abierta, que pueda ser fácilmente modificada y extendida sin ningún tipo de impedimento por licencias restrictivas.

Como resultado de la consecución de estos objetivos nos encontramos con una arquitectura totalmente nueva de *Marte*, caracterizada por el modelo cliente/servidor. El servidor utilizado será abierto y de carácter centralizado por el que pasará cada flujo de información multimedia de las conferencias,

[1] La última noticia en la portada de su página web data de Noviembre de 2004.

aunque tiene como único punto en contra que el protocolo utilizado (RTMP) es propietario. Este servidor se utilizará como indicador de presencia de los clientes, *proxy* de las sesiones VNC, servidor de autenticación, y distribuidor de los datos de vídeo y audio. La parte cliente será una aplicación del mundo de la *web* 2.0, ejecutable en la mayoría de navegadores *web* y de sistemas operativos, con la idea de ser un cliente ligero y muy sencillo de utilizar aprovechando las posibilidades de *Flex* para crear interfaces de usuario.

B. Arquitectura del servidor

La arquitectura del servidor está basada en *Red5*, un servidor *Flash* de código abierto que nos permite, entre otras cosas, distribuir contenidos multimedia a través del protocolo RTMP o de su variante sobre HTTP, RTMPT.

La elección de *Red5* sobre los otros servidores disponibles para esta plataforma (*Adobe Media Server*, *Wowza*) se debe sobre todo a que se trata de un proyecto de código abierto escrito en Java lo que permite, por un lado, escribir aplicaciones de servidor en este lenguaje y además, modificar el código fuente del propio servidor para realizar tareas más específicas imposibles en las otras opciones disponibles.

1) Planificación en Red5

Red5 se sostiene en la estructura MINA (*Multi-purpose Infrastructure for Network Applications* [13]) de Apache que permite desarrollar aplicaciones de red altamente escalables basándose en parte en la arquitectura SEDA [3] (*Staged Event-Driven Architecture*).

SEDA propone descomponer la aplicación de red en un modelo de etapas conectadas por colas que permite filtrar el tráfico de entrada en cada una de dichas colas, acondicionando de esta manera el flujo entrante para mejorar el rendimiento en los picos de tráfico ya que todo puede ser ajustado dinámicamente. Cada una de estas etapas es una pieza del software independiente que realiza parte del procesado de la petición pasándolo, en cada caso, a la cola siguiente que corresponda, todo esto como alternativa a la proliferación de hilos que añaden más sobrecarga y que no permiten reaccionar tan fácilmente ante un pico en un momento determinado.

El ya anteriormente mencionado MINA es parte del proyecto Apache e intenta aprovechar la filosofía descrita por SEDA para facilitar la tarea de realizar este tipo de aplicaciones en Java y la separación entre la gestión de eventos más implementación de protocolos de la lógica de la aplicación propiamente dicha.

2) Protocolos

Como se menciona anteriormente, *Red5* utiliza el protocolo RTMP (o RTMPT) para la distribución de contenidos multimedia. El objetivo de este protocolo es que los clientes hechos con la tecnología *Flash* (en su momento también de Macromedia) pudieran enviar y recibir datos en tiempo real con un mínimo de garantía desde y hacia el servidor *Flash Media Server*. Una de las características que permite esto es la posibilidad de tunelar el tráfico para que los datos viajen como si fueran el cuerpo de mensajes HTTP, de forma que no sean tan fácilmente visibles en dispositivos como *firewalls* y gestores de tráfico que son configurados en la frontera de las redes empresariales y personales para filtrar contenidos de este tipo. Esta última característica facilita enormemente el uso del servicio de videoconferencia en ambientes muy restringidos sin, en principio, tener que implementar cambios importantes en el cliente ni en el servidor.

3) Aplicaciones del servidor

Red5 utiliza un servidor HTTP y de aplicaciones (*servlets*) *Jetty* [14] administrado por la estructura *Spring* que pone a disposición del programador la posibilidad de configurar de diversas maneras (XML, archivos de propiedades Java,...) denominadas "ganchos" el comportamiento de diversas aplicaciones Java. De esta manera es posible codificar en Java una aplicación de servidor que corra sobre Red5 utilizando el interfaz ofrecido y configurar diversos parámetros editando varios archivos XML.

En lo que concierne a *Marte* 3.0 se ha aprovechado las posibilidades que ofrece *Red5* para crear nuevas aplicaciones de servidor y se han implementado las partes necesarias para cumplir con los requisitos expuestos por las versiones anteriores de Marte con ciertas variaciones en algunos casos para adaptarse a la nueva situación.

En primer lugar, la información de los usuarios sigue estando almacenada en un LDAP, siguiendo exactamente el mismo formato que en la versión anterior, permitiendo una compatibilidad total en este sentido, y realizándose la autenticación de los clientes de manera muy similar.

Además, es imprescindible que el servidor mantenga información sobre la presencia de los clientes conectados, estén o no participando activamente en alguna conferencia. Para mantener la capacidad conferencia multimedia existente en *Marte* 2.0, se ha implementado un sistema de habitaciones, representando cada una de ellas una conferencia independiente, pudiendo cada cliente participar únicamente en una en cada momento. Asociada a esta nueva estructura de habitaciones, se ofrece a los clientes la posibilidad de invitar a otros a una habitación determinada.

Por otro lado, el servidor mantiene información sobre las capacidades de cada cliente, es decir, si puede (y quiere) emitir vídeo y audio (tiene una cámara conectada) y/o de compartir su escritorio mediante VNC. Esta información es ofrecida al resto de los clientes que podrán así elegir en cada momento los flujos que reciben.

Por consistencia con la versión anterior de *Marte*, siguen existiendo los modos de interacción, siendo necesario para su funcionamiento que el servidor informe a los clientes del modo activado actualmente, encargándose estos de organizar el interfaz de la manera adecuada como se explicará posteriormente en la sección sobre la arquitectura del cliente.

La mayor parte de esta comunicación se realiza mediante llamadas a métodos remotos desde los clientes aprovechando las capacidades para ello ofrecidas por *Red5*, las respuestas del

servidor son enviadas como objetos serializados e interpretadas por el cliente como sea conveniente.

4) Arquitectura de escritorio compartido

Debido a las limitaciones impuestas por la tecnología usada en el cliente que se detallarán mas adelante, es necesario que los clientes de escritorio compartido se conecten a la misma dirección en la que está desplegado el servidor de *Red5*, esto obliga a pensar en una arquitectura diferente de la habitual, en la que los clientes se conectarían directamente al usuario que decidiese compartir su escritorio en cada momento.

Entre las alternativas, se optó por utilizar VNC *Reflector* [16] en el servidor. VNC *Reflector* es un servidor especial de VNC que actúa como *proxy* entre un servidor de VNC y los clientes, evitando efectivamente el problema planteado, ya que los clientes pasan a conectarse, a la hora de compartir el escritorio, directamente a la misma dirección en la que se encuentra el *Red5*. Esto obliga a crear una capa entre la aplicación de servidor existente en *Red5* y dicho VNC *Reflector* para arrancarlo con los parámetros adecuados (los puertos de a los que deben conectarse el cliente que comparte su escritorio y el resto) e informar sobre los servidores existentes y las correspondencias necesarias, esta capa actúa como indica la ¡Error! No se encuentra el origen de la referencia..

Como resultado final, el *applet* de Java que hace las veces de servidor VNC obtiene la dirección y el puerto al que debe conectarse (los del VNC reflector) y el módulo FVNC del resto de participantes se conecta de manera transparente para usuario permitiendo, de mantera efectiva el uso del escritorio compartido.

C. Arquitectura del cliente

1) Aplicaciones ricas de Internet

El concepto de aplicaciones ricas de *Internet* fue descrito en [4], y actualmente está teniendo mucho auge en *Internet*. Según este artículo una tecnología de cliente rica debe cumplir con los siguientes aspectos: debe proveer un entorno de ejecución eficiente que permita además de ejecutar el código, tratar contenidos y comunicar el cliente con aplicaciones externas, debe integrar en un entorno común estos contenidos, comunicaciones e interfaces (no como el actual HTML), permitir que el usuario interactúe con modelos de objetos extensibles (mejorando lo que nos proporciona *Java Script* y DHTML), facilitar un desarrollo rápido de las aplicaciones a través del uso de componentes reutilizables, comunicarse con servidores de aplicaciones a través de los servicios de datos y de la *web*, permitir a las aplicaciones ejecutarse en los estados conectado y desconectado, y ser fácilmente desarrollable en múltiples plataformas y dispositivos. El objetivo final es obtener aplicaciones que se ejecutan en un navegador *web*, pero que mantienen las mismas características y funcionalidad que las aplicaciones de escritorio.

Las ventajas de este tipo de desarrollo son que las aplicaciones resultantes en general no requieren instalación ya que se ejecutan en los propios navegadores *web* a través de pequeñas aplicaciones instalables (*plug-in*), y que con el uso de éstos se ejecutan localmente bajo la supervisión de un entorno de seguridad comúnmente denominado cajón de arena (*sandbox*). Además las actualizaciones del software son automáticas ya que los usuarios se descargan de la *web* directamente la última versión y la versión normalmente es ejecutable en un gran número de máquinas sin importar el Sistema Operativo y el navegador que las utiliza. Por tradición este tipo de aplicaciones siempre han mejorado la interfaz de usuario acercándose a lo que podemos ver en las aplicaciones de escritorio, gracias a que utilizan características tales como arrastrar y soltar, barras de desplazamiento, vídeo, audio, gráficos vectoriales con transformaciones, efectos de sombras, etc. El hecho de hacer uso de peticiones a un servidor de forma asíncrona permite balancear la carga de procesamiento entre el cliente y el servidor y usar la red de forma más eficiente.

A pesar de que estas aplicaciones surgen como respuesta a una demanda real por parte de los usuarios existen también

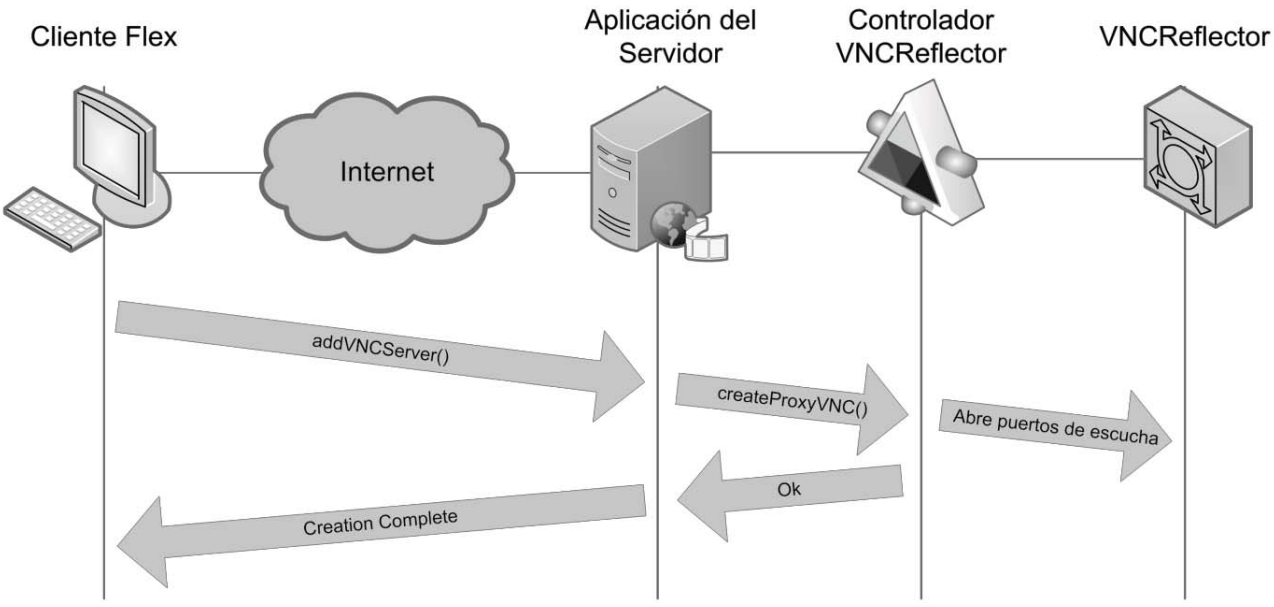

Fig. 3 Escenario de control de flujos VNC

desventajas por su uso, comenzando por la propia utilización del *sandbox* de seguridad (que restringen el acceso a los recursos), el que un usuario tenga que descargar la aplicación *Web* hace que tenga que esperar un tiempo adicional para poder ejecutarla, la dependencia de una conexión de *Internet*, el hecho de que el desarrollo de una aplicación *web* no es igual que el de una aplicación de escritorio y en muchos casos hay que rehacer las aplicaciones desde cero, y por último que las aplicaciones ricas de *Internet* (RIA) están aún en un fase muy temprana de adopción por lo que no todas las tecnologías han sido acogidas por todos los navegadores *web*.

Los marcos de desarrollo de RIA más utilizados en la actualidad son AJAX [17] (que ha sido acogido por empresas como Google y *Yahoo* para crear clientes que consultan sus propios servicios), *JavaFX* [18] (de *Sun Microsystems*), los *applets* de Java [19] (aunque su desarrollo es más complejo), *Microsoft Silverlight* [20] (que es la apuesta de Microsoft en el mundo de las RIAs), y el más utilizado hasta la fecha Adobe *Flash*/*Flex*. En nuestro caso optamos por utilizar Adobe *Flex*, ya que es un marco de desarrollo de código abierto que permite crear flujos de vídeo y audio desde el navegador de forma muy sencilla, tanto para el desarrollador como para los usuarios finales.

2) Interfaces de usuario

Nuestra aplicación se desarrolló en *Flex* con código *ActionScript* y MXML. *ActionScript* es un lenguaje de programación orientado a objetos similar a Java. Lo creó Macromedia (actualmente Adobe) para generar, una vez compilado, código *Flash* ejecutable (que se denomina SWF) por el reproductor de la misma compañía. Este lenguaje está basado en la cuarta edición de la especificación de *ECMAScript* (que aún no se ha publicado una versión definitiva, pero extiende de la tercera versión) y una de sus principales características es la adopción de otro estándar ECMA para codificar documentos XML como objetos del mismo lenguaje.

El lenguaje MXML sirve para definir interfaces de usuario avanzadas en formato XML. Una vez que el desarrollador ha creado la interfaz con MXML el compilador *Flex* traduce el documento en las clases *ActionScript* necesarias para después compilarlo.

Por lo tanto la aplicación de *Marte* diferencia la interfaz de usuario del resto de lógica de la aplicación, que explicaremos en el próximo apartado con más detalle. En cuanto a la interfaz de usuario se han añadido tres estados diferentes de la aplicación: Sin conexión, conexión a la sala principal y conexión a una de las salas de videoconferencia. Cada uno de los tres estados muestra diferentes configuraciones de ventanas y posibilidades al usuario. Aprovechando el potencial de *Flex* para el diseño dinámico de efectos se crearon transiciones con efectos entre cada uno de los estados.

Además una vez conectado a una sala un usuario puede estar dentro de otro tipo de estados que en *Marte* se ha venido definiendo como modo de interacción. Estos modos en *Marte* 3.0 son los siguientes:

- Modo N+1: La aplicación muestra al usuario el vídeo de uno de los participantes en la conferencia en grande mientras que el resto de participantes se pueden ver en pantallas más pequeñas. Este modo se diseñó en la primera versión de Marte para conferencias en las que uno de los participantes habla sobre un tema y los demás escuchan.

- Modo chat: El usuario puede ver los vídeos de los demás participantes, todos con el mismo tamaño en pantalla. Este modo se diseñó para ocasiones en las que las conferencias son reuniones en las que todos los usuarios participan igualmente.

- Modo *One*: Sólo se ve el vídeo del participante que va a hablar. Sirve para conferencias tipo ponencias, en las que no se espera que ningún otro asistente intervenga en la charla.

- Modo VNC: En grande el interfaz muestra un escritorio compartido y todos los vídeos de la conferencia. Está pensado para conferencias en las que va a haber exposiciones acompañadas de conferencias o demostraciones en una de las máquinas.

3) Arquitectura del cliente Flex

La arquitectura del cliente, que podemos ver en la Fig. 4 se puede dividir en cuatro módulos que se diferencian en el propósito de cada uno de ellos: Módulo del control de sesión, de control de los flujos multimedia y del escritorio compartido.

El primero de ellos es el que contempla todas las clases encargadas de controlar los aspectos básicos de la sesión. La clase principal de este módulo es el controlador de sesión, que es el encargado de enviar y recibir mensajes al servidor *Red5* manteniendo el estado de sesión y datos sobre los usuarios conectados, las salas disponibles y el modo de presentación que hay en la sala en la que el usuario está conectado.

El control de los flujos multimedia se hace utilizando la clase *NetStream*, que pertenece al API que ofrece el marco de desarrollo *Flex*. Por encima utilizamos objetos avanzados que nos permiten controlar parámetros de los flujos como el volumen del audio, el aspecto de la imagen, etc. Estos dos primeros módulos se apoyan en otra clase de *Flex* llamada *NetConnection*, que es la clase especializada en realizar conexiones RTMP/RTMPT hacia servidores multimedia.

El módulo del escritorio compartido se divide en dos partes: una es la implementación del cliente VNC que es producto de un proyecto de software libre llamado *FlashVNC*; la biblioteca es un cliente completo de VNC para la versión 3.3 del protocolo. Éste permite recibir la pantalla del escritorio compartido y enviar eventos del ratón y del teclado al servidor VNC. La segunda parte es la que hace de servidor VNC, que la hemos realizado con tecnología Java por los problemas que explicamos

en el siguiente apartado.

Por encima de cada módulo se han creado interfaces definidas en MXML para mostrar al usuario la información necesaria y recibir las órdenes de éste por medio de eventos del teclado y ratón.

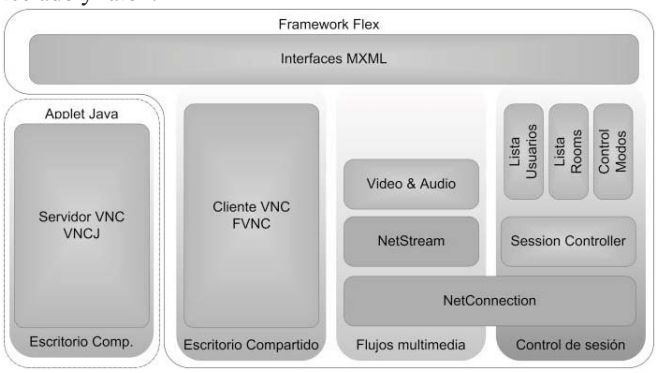

Fig. 4 Módulos de desarrollo en el cliente Marte

4) Problemas de compartición de escritorio

Como solución encontramos dos bibliotecas de código abierto que implementan la parte cliente y servidora de VNC: Para la parte cliente utilizamos FVNC, que fue desarrollada en *ActionScript* para poder incluir clientes de escritorio compartido dentro de aplicaciones *Flash/Flex*; su utilización es muy sencilla ya que casi no obliga a modificar el código para implementaciones propias. Sus características se basan en que utiliza la versión 3.3 de RFB [8], que es la más antigua de las que se pueden ver por *Internet*, pero no por ello peor.

En la parte servidora los problemas encontrados fueron derivados de las políticas de seguridad sobre las que se rige la máquina virtual de *Flash*. Entre estas políticas destacan que no es posible abrir puertos TCP que atiendan a intentos de conexión externos y que no es posible realizar capturas de la pantalla de la máquina en la que se está ejecutando la aplicación. Por otra parte estas dos características son obvias, ya que esta tecnología está pensada para ser ejecutada en el navegador *web* del cliente, por lo que la seguridad es un aspecto muy importante. La solución para el servidor VNC la encontramos en el uso de *applets* Java, con los que sí que se pueden aceptar conexiones y realizar capturas de pantalla siempre que se firme digitalmente el *applet* que se ejecutará en el navegador del cliente. El resultado final es el que veíamos en la figura anterior.

IV. Conclusiones y trabajos futuros

Como comentamos en la introducción el objetivo final de esta implementación era obtener un sistema de videoconferencia con un diseño fácil, con una interfaz sencilla para el usuario final y que pudiera ser utilizado en la mayoría de sistemas operativos sin necesidad de instalaciones complejas. Durante el estudio inicial comprobamos que la mejor herramienta para lograrlo era la tecnología *Flex*, por que ha sido creada por una empresa dedicada históricamente a la fabricación de sencillas herramientas de diseño *web* (antes como *Macromedia* y ahora como *Adobe*). Hemos visto como uno de los principales compromisos de Adobe, que es la seguridad en este tipo de aplicaciones, nos ha impedido desarrollar ciertas funcionalidades como la compartición de escritorio por lo que tuvimos que acudir a la tecnología Java, por lo que el resultado ha sido el que buscamos inicialmente. Comparando los sistemas anteriores con la nueva versión, comprobamos que la utilización de *Adobe Flash* presenta, por el momento, una ligera pérdida de calidad de video, pero que se ve ampliamente compensada por el hecho de no tener que instalar ningún tipo de aplicación y poder accede desde cualquier rincón de *Internet*.

Por ello este proyecto es un buen punto de partida para futuras investigaciones que busquen objetivos similares. El punto en contra es que en cuanto la complejidad aumenta y se quiere seguir teniendo compatibilidad con otros sistemas surgen problemas que hacen que el desarrollador se tenga que plantear el resolverlo utilizando otras tecnologías de la *web* 2.0 o incluso fuera de este escenario.

En la Fig. 5 vemos las posibilidades que ofrece una arquitectura de este tipo para su utilización desde dispositivos móviles, demostrando que es posible conseguir la conexión desde un amplio conjunto de dispositivos.

Con los resultados obtenidos se han iniciado investigaciones para resolver temas más concretos del proyecto, como pueden ser la mejora del servicio de escritorio compartido para que viaje con el resto de información dentro de peticiones HTTP. O la inclusión de nuevos servicios como son la pizarra compartida o la mensajería instantánea. Otro punto de interés sería el aumento de compatibilidad con otros sistemas de videoconferencia presentes en *Internet*, como pueden ser *Jingle*, SIP, etc. Además la facilidad de creación de interfaces de *Flex* hace que sea posible la búsqueda de nuevos modelos de interfaz basados en la *web* 2.0 para el control de conferencias de vídeo. Por último también se está trabajando actualmente en la creación de distintos tipos de interfaces para la gestión y consulta de videoconferencias, como son los servicios web (*Web Services*) y *Rest*.

Fig. 5 Ejemplo de funcionamiento con dispositivos móviles

REFERENCIAS

[1] D. Machín Vázquez-Villa, "Diseño de un entorno de servicios de colaboración multiusuario basado en SIP", Proyecto Fin de Carrera dirigido por T. de Miguel Moro, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Diciembre de 2004.

[2] M. Gómez Rodríguez, "Contribución a la provisión de servicios de colaboración de nueva generación en redes IMS", Tesis doctoral dirigida por T. de Miguel Moro, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Abril de 2008

[3] M. Welsh, "An Architecture for Highly Concurrent, Well-Conditioned Internet Services" Ph.D. Thesis, University of California, Berkeley, August 2002.

[4] J. Allaire, "Macromedia Flash MX-A next-generation rich client" Macromedia, March 2002. Disponible en: http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf . Última visita: 06/06/2008

[5] "ECMAScript Language Specification" Estándar ECMA. December 1999.

[6] John Schneider, "ECMAScript for XML (E4X) Specification" ECMA Standard, December 2005. Disponible en: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf . Última visita: 06/06/2008

[7] Richardson T. Stanford Q. "Virtual Network Computing". IEEE Internet Computing, Vol2, No 1 January/February 1998

[8] Tristan R, et al (2002). "The RFB Protocol". Revision 2007. Disponible en: http://www.realvnc.com/docs/rfbproto.pdf . Última visita: 06/06/2008

[9] C. Coenraets, "An overview of MXML: The Flex markup language", Adobe Systems. March 2004. Disponible en: http://www.adobe.com/devnet/flex/articles/paradigm.html Última visita: 06/06/2008

[10] Java Media Framework: http://java.sun.com/products/java-media/jmf/ . Última visita: 06/06/2008

[11] Adobe Flash: http://www.adobe.com/go/gntray_prod_flash_home_es . Última visita: 06/06/2008

[12] Real Time Messaging Protocol: http://www.adobe.com/go/tn_16631 . Última visita: 06/06/2008

[13] MINA: http://mina.apache.org/ .Última visita: 06/06/2008

[14] Jetty Server: http://www.mortbay.org/ .Última visita: 06/06/2008

[15] Spring Framework: http://www.springframework.org/ .Última visita: 06/06/2008

[16] VNCReflector: http://sourceforge.net/projects/vnc-reflector/ .Última visita: 06/06/2008

[17] Asynchronous JavaScript And XML : http://www.adaptivepath.com/ideas/essays/archives/000385.php Última visita: 06/06/2008

[18] JavaFX: http://sun.com/javafx .Última visita: 06/06/2008

[19] Java Applets: http://java.sun.com/applets/ .Última visita: 06/06/2008

[20] Microsoft Silverlight: http://silverlight.net/ .Última visita: 06/06/2008