Servicios Web para Rendering de Volúmenes No Estructurados

Mauricio López, Elsa Estévez, Silvia Castro, Sergio Martig

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, Argentina
maur.lopez@gmail.com
{ece, smc, srm}@cs.uns.edu.ar

Abstract

The visualization of unstructured tetrahedral meshes has been proven to be costly in terms of computational resources and it can easily overcome the local computer's capacity. In this work we describe the general design and implementation details of a remote visualization service able to deal with unstructured volumetric data. Specifically we focus on exposing the rendering process as a Web Service. We analyze different interactions between client applications and the service, and the entire data flow received and sent by the service as SOAP messages. Moreover, we provide an interactive graphical user interface for filling the leak of interactivity of Web Services, showing the feasibility of this architecture.

Keywords: Scientific Visualization, Web Services, Unstructured Meshes, Volume Rendering, SOAP.

Resumen

La visualización de mallas tetraédricas no estructuradas ha probado ser costosa en términos computacionales y puede fácilmente sobrepasar la capacidad de procesamiento local. En este trabajo describimos el diseño general así como los detalles de implementación de un servicio de visualización remota capaz de procesar datos volumétricos no estructurados. Concretamente nos centramos en la exposición del proceso de rendering como Servicio Web. Analizamos las distintas interacciones entre las aplicaciones cliente y el servicio y todos los flujos de datos recibidos y enviados por el servicio como mensajes SOAP. Asimismo, proveemos al usuario de una interfase visual que suple la falta de interactividad de los Servicios Web mostrando así la factibilidad de esta arquitectura.

Palabras Clave: Visualización Científica, Servicios Web, Volúmenes no Estructurados, Rendering de Volúmenes, SOAP.

1 INTRODUCCION

La computación realizada sobre la GRID tiene cada vez más relevancia para la comunidad de visualización ya que extiende sus horizontes y posibilidades tanto desde el punto de vista de nuevas funcionalidades, como así también de procesamiento distribuido. Los Servicios Web representan una herramienta para construir la GRID aportando la interoperabilidad necesaria entre plataformas, así como la independencia de lenguajes de programación. Por definición [15], un Servicio Web es un sistema de software diseñado para soportar interacciones entre computadoras conectadas en red, con una interfase descripta en un lenguaje interpretable por máquinas (específicamente WSDL –

Web Services Description Language [21]) y donde otros servicios pueden interactuar con él mediante mensajes SOAP – Simple Object Access Protocol [22].

Actualmente, uno de los desafíos para la comunidad de visualización es proveer sistemas de visualización que estén naturalmente integrados a la computación en la GRID. Uno de los objetivos de estos sistemas es que el usuario pueda visualizar enormes cantidades de datos desde un computador común con acceso a Internet. Por ejemplo, el Rendering Directo de Volúmenes representa un caso típico de visualización que requiere gran cantidad de recursos computacionales. Una arquitectura de sistemas orientada a servicios (Service-Oriented Architecture , SOA [23]) provee una solución adecuada y eficiente para este tipo de sistemas. En este enfoque, el proceso de rendering puede ser expuesto como un Servicio Web, que puede ser ejecutado por un servidor especializado – sin importar su ubicación geográfica, y accedido por múltiples usuarios.

La Visualización Directa de Volúmenes en la Web ha sido tema de investigación en los últimos años. Si bien la mayoría de los enfoques plantean sistemas interactivos, solo toman en cuenta la visualización de volúmenes regulares y en especial aquellos que poseen una cantidad moderada de voxels. En su mayoría, tales volúmenes de datos son visualizados en la máquina local. Sin embargo, la visualización en la Web de volúmenes realmente grandes no estructurados no ha recibido la atención adecuada como tema de investigación.

Este trabajo describe el diseño general, las posibles interacciones y las tecnologías utilizadas en la implementación de una arquitectura de software orientada a servicios para Rendering de Volúmenes - en especial para volúmenes no estructurados. En la sección siguiente se describen los trabajos relacionados en el área de sistemas de visualización distribuidos y de servicios web. En la Sección 3 se presenta la arquitectura de la solución propuesta detallando las tecnologías utilizadas para su implementación. Adicionalmente, se explican las posibles interacciones entre la aplicación cliente y los servicios web, y entre el usuario y la aplicación cliente. Finalmente, en la cuarta sección se presentan las conclusiones.

2 TRABAJOS RELACIONADOS

A continuación se describen los antecedentes relacionados con las dos áreas principales de investigación del presente trabajo: los sistemas de visualización y los servicios webs.

2.1 Sistemas de Visualización

En 1987, McCormick, de Fanti y Brown [10] proponen los fundamentos para la visualización distribuida, sugiriendo el uso de una red de supercomputadoras para ejecutar cálculos científicos y de estaciones de trabajo para realizar el rendering. McCormick [10] plantea los Ambientes de Visualización Modular (MVE - Modular Visualization Environments), formados por módulos independientes que realizan tareas sobre los datos realizando algún tipo de transformación. Típicamente los datos pasan de un módulo a otro siguiendo un patrón de flujo de datos. El conjunto de módulos incluye un módulo inicial responsable de la lectura de los datos y uno final encargado de realizar el rendering. El conjunto puede contener varios módulos, cada uno con una funcionalidad específica; por ejemplo, un módulo responsable de extraer una iso-superficie. La ejecución de los módulos puede realizarse en secuencia o en paralelo. En el caso que los módulos se ejecuten en paralelo, resulta sencilla su implementación en base a un modelo distribuido. Este modelo incluye un sofisticado editor gráfico que permite construir las aplicaciones seleccionando

los módulos a utilizar y cómo se enlazarán. De este modo el usuario crea su propia línea de producción o pipe line. Ejemplos de estas aplicaciones son IRIS Explorer [17] e IBM Data Explorer [18]. La Figura1 ilustra el enlace de tres módulos para visualización de iso-superficies poligonales ejecutados secuencialmente, y las correspondientes interacciones con el cliente.

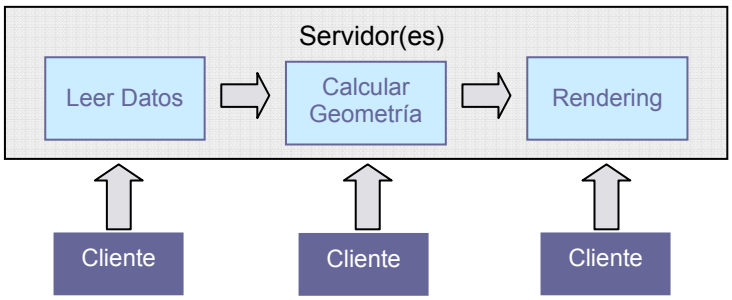

Figura 1: Visualización de Iso-Superficies como Flujo de Datos (McCormick)

El modelo de flujo de datos continúa utilizándose y generalmente es la base de todos los sistemas de visualización distribuida así como de varios sistemas stand alone.

En 1994, Ang et al. [1] propuso un sistema de visualización basado en la Web, utilizando la misma solo para recoger datos remotos. En este enfoque, se copian los datos de la Web a la máquina cliente transportándolos como archivos adjuntos de tipo MIME. La visualización se hace usando el poder de procesamiento de la máquina cliente. Posteriormente Yeo [14] extendió este enfoque, proponiendo que algunos módulos sean ejecutados en el servidor antes de enviar los datos al cliente. Yeo utilizó IRIS Explorer para organizar los módulos y el flujo de datos. En esta solución, se requiere transmitir al cliente una serie de scripts que contienen la configuración de IRIS - información del enlace de módulos, lo cual se realiza enviando esta información como archivos adjuntos MIME. En esta arquitectura, el rendering final se realiza en el cliente.

Mas recientemente, en 2000, Engel [6, 7] plantea un sistema de visualización mas dinámico basado en Applets de Java. El usuario descarga de la Web la aplicación cliente, transmite los datos al servidor vía FTP, éste procesa los datos y retorna un archivo VRML (Virtual Reality Modeling Language [16]) para que el rendering se realice localmente por el navegador. Asimismo, se contempla que los datos residan en el servidor. En ese caso, el Applet solo envía ciertos parámetros al servidor y éste retorna al cliente un archivo VRML para realizar la visualización en forma local.

Un enfoque reciente, es el Servicio Web de “Visible Human Project” ofrecido por EPFL (Ecole Polytechnique Fédérale de Lausanne) [19]. Muestra un Applet como interfase para seleccionar slices o “cortes” de interés del dataset. En esta arquitectura, el pedido es procesado por los servidores, los cuales extraen las porciones de datos necesarias y las retornan al Applet para que ejecute la visualización de manera local.

Las soluciones propuestas por Engel, Ang y Yeo asumen que el cliente tiene poder de procesamiento para realizar el rendering - asumen que la máquina cliente cuenta con suficientes recursos de memoria y espacio de almacenamiento secundario. El poder de procesamiento de la máquina cliente es un requerimiento crucial en estos enfoques, ya que los mismos permiten realizar la visualización en forma interactiva. De este modo, el usuario puede ver inmediatamente el resultado de sus interacciones con el sistema. Estos enfoques no contemplan el caso de que el cliente no posea capacidad de procesamiento.

Thiesel [3] y otros investigadores proponen que la visualización se haga del lado del servidor pero dirigida por el cliente. Es decir, los datos y los módulos que conforman el modelo de flujo de datos residen en el servidor; el cliente planea y dirige la visualización y recibe como resultado un mapa de bits. Thiesel en su sistema CurVis [3] produce visualizaciones de dinámica de fluidos 2D y envía una imagen comprimida al cliente móvil. Yunson Wang [13] propone una alternativa que consiste en derivar el rendering final al cliente, solo si éste tiene poder de procesamiento; en caso contrario, le envía solo una imagen, con lo cual, el cliente pierde interactividad.

En base a los trabajos mencionados, se concluye que la visualización ejecutada del lado del servidor parece adecuada para visualizaciones de datos cuyo volumen sobrepasa la capacidad de procesamiento local - como por ejemplo el rendering de volúmenes no estructurados. En estos casos, si bien es factible transmitir los datos al servidor para ser visualizados, es más probable que los datos residan en los servidores, ya que muchas veces son generados por medio de una simulación o por métodos numéricos, como por ejemplo las mallas generadas por FEM (Finite Element Method).

2.2 Servicios Web

En la última década, W3C Consortium, Organization for the Advancement of Structured Information (OASIS) y varias empresas privadas han promovido la adopción de varios estándares que han dado lugar al nacimiento de una infraestructura común de mensajería. Los Servicios Web se basan en estándares (XML, XML-Schema, SOAP, WSDL y UDDI) y no dependen de ninguna plataforma, lenguaje o hardware en particular. Los clientes de un Servicio Web pueden ser ejecutados sobre PDAs, laptops, teléfonos celulares, PCs, mainframes, etc., y pueden estar escritos en diferentes lenguajes de programación.

Conceptualmente, un Servicio Web es un conjunto de funciones o procesos que se ejecutan en un servidor remoto. Para la aplicación cliente, la ejecución de un Servicio Web es transparente, ya que los servicios son invocados por nombre - al estilo de las llamadas a procedimientos remotos, dejando a un servidor proxy que se ocupe de los detalles de invocación. El cliente se conecta a los servicios vía un acceso punto a punto conociendo el URL/IP de destino. Esta información puede conocerse durante la etapa de diseño y ser estática para el sistema, o puede obtenerse en tiempo de ejecución - en forma dinámica, haciendo uso de registros de servicios, de acuerdo al framework definido por UDDI – Universal Description, Discovery and Integration [24]. UDDI provee al cliente el URL/IP del servicio y una descripción del mismo escrita en WSDL.

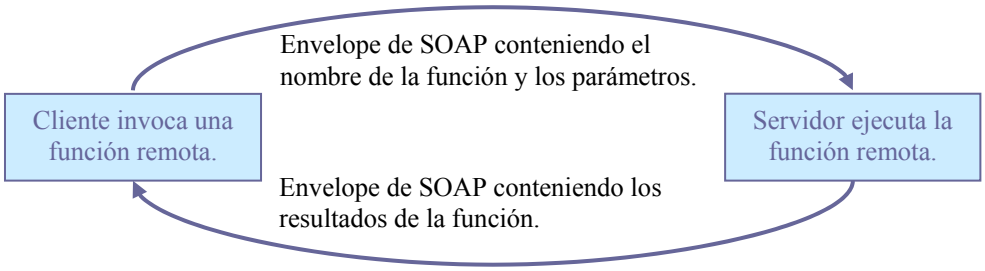

Figura 2: Mensajes Recibido y Enviado por un Servicio

Generalmente los desarrolladores escriben proxies, los cuales son clases o wrappers que crean los envoltorios de SOAP y se encargan de toda la mensajería (ver Figura2). El Framework .Net de Microsoft permite generar estos proxies de manera automática a partir de la descripción del servicio

en WSDL. La misma funcionalidad es ofrecida por AXIS, el motor de SOAP propuesto por Apache. De este modo cualquier modificación en la estructura de mensajes solo afecta al proxy y no a las aplicaciones cliente y servidor. Una vez que el cliente descubre el servicio en un registro, el cliente invoca directamente al Servicio Web mediante su Proxy.

3 Arquitectura para Rendering de Volúmenes No Estructurados

La arquitectura de software propuesta para rendering de volúmenes no estructurados es una arquitectura orientada a servicios. La misma incluye, conceptualmente, dos servicios web: uno es responsable de gestionar los datos a visualizar - *Servicio de Datos*, el otro de realizar el rendering - *Servicio de Rendering*. Asumimos que los conjuntos de datos están listos para ser visualizados, que están presentes todos los datos visualizables, y poseen mapeo visual, en nuestro caso, *Función de Transferencia*. La Función de Transferencia es una función continua multivaluada $f(s) = (c, \alpha)$ cuyo dominio *s* son todos los escalares del volumen (una señal continua) y cuyo rango es una serie de colores *c* y transparencias α, generalmente empaquetados en números de 32 bits (RGBA). En otras palabras, se trata de un *mapeo visual* que traslada los escalares del dominio físico original hacia el dominio espectral de colores y transparencias. En la práctica notamos que varios voxels comparten el mismo color y otros tantos la misma transparencia, por lo tanto resulta *práctico* crear una paleta, relativamente pequeña, con aquellos colores y transparencias y guardarla convenientemente como un archivo comprimido JPG.

Las siguientes dos secciones explican los casos de uso soportados por la arquitectura.

3.1 Casos de Uso entre Aplicación Cliente y Servicios Web

A fin de poder realizar la visualización, el cliente deberá primero seleccionar los datos a visualizar. En caso que los datos no estén disponibles en el servidor, el cliente podrá subir un archivo. Una vez obtenidos los datos se podrá realizar una vista previa interactiva de los mismos basada en la superficie del volumen o bien en el convex hull. La Figura 3 muestra el diagrama de casos de uso que incluye estas funcionalidades, las cuales se detallan a continuación:

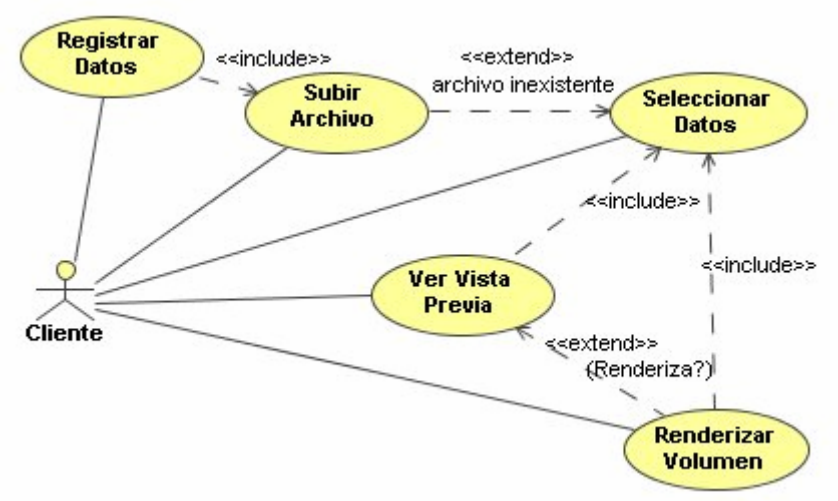

Figura 3: Diagrama de Casos de Uso – Aplicación Cliente y Servicios Web

1) *Registrar Datos*: todo conjunto de datos que se desee visualizar, deberá ser previamente registrado. El proceso de registración permite que el cliente informe la ubicación física y nombre del archivo conteniendo los datos, y una serie de calificadores para facilitar la búsqueda del conjunto. El proceso asigna un identificador único a cada conjunto. Debido a que cada conjunto de datos tiene anexada una o varias funciones de transferencia, éstas son registradas en una tabla secundaria cuya clave principal está compuesta por el identificador del conjunto de datos al que pertenece (clave foránea) y un identificador autogenerado.
2) *Subir Archivo*: en caso que el archivo resida en la maquina cliente, automáticamente se solicita subir el archivo al servidor.
3) *Seleccionar Datos*: dos tipos de búsqueda se ofrecen para seleccionar el conjunto de datos: el criterio de búsqueda en páginas blancas – búsqueda por identificadores, y el de páginas amarillas – búsqueda de acuerdo a los calificadores informados durante la registración.
4) *Ver Vista Previa*: considerando que la visualización es costosa en términos computacionales y en consecuencia no es interactiva, es deseable que el usuario pueda contar con una pre-visualización interactiva que lo asista. En este caso la aplicación cliente invoca la función de ver vista previa especificando el identificador del conjunto de datos. El servicio Vista Previa genera un archivo con geometría sencilla y lo envía al cliente para que se muestre una interfase visual interactiva y el usuario pueda definir parámetros de visión - como por ejemplo situar la cámara.
5) *Renderizar Volumen*: el usuario solicita la visualización de un conjunto de datos indicando el identificador del conjunto de datos, el identificador de la función de transferencia y los parámetros de visión. Con estos datos, la aplicación cliente invoca al servicio *Rendering Volumen* el cual genera un mapa de bits, lo comprime en formato JPG y lo envía como respuesta al cliente.

3.2 Casos de Uso entre el Usuario y la Aplicación Cliente

El usuario interactúa con la aplicación cliente a fin de seleccionar el dataset a visualizar y luego durante la visualización misma. Los casos de uso definidos para seleccionar el conjunto de datos se muestran en la Figura 4 y se describe a continuación:

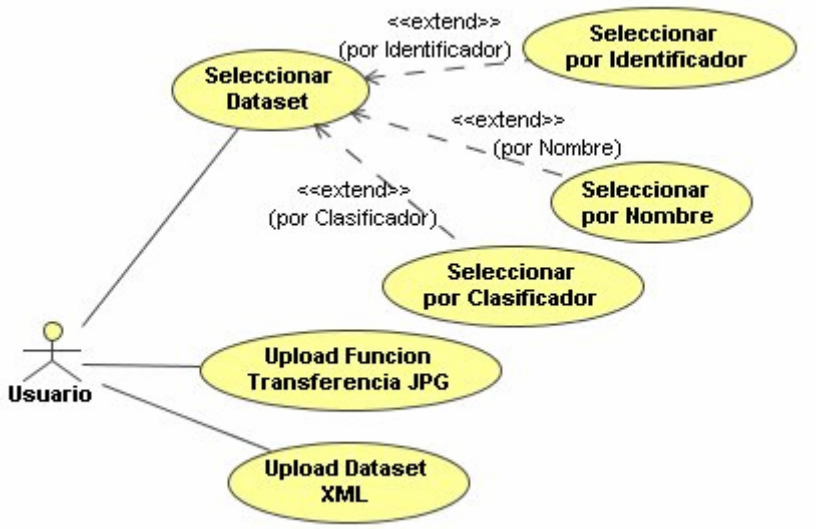

Figura 4: Diagrama de Casos de Uso - Interacciones del Usuario con el Servicio de Datos

1) *Seleccionar Dataset*: permite que el usuario seleccione el dataset a visualizar. Este caso de uso es extendido por los que se mencionan a continuación dependiendo del criterio de selección.

2) *Seleccionar por Identificador*: permite seleccionar el dataset ingresando el identificador.
3) *Seleccionar por Nombre*: permite seleccionar el dataset informando el nombre.
4) *Seleccionar por Clasificador*: selecciona todos aquellos dataset que poseen el mismo calificador que el informado por el usuario.
5) *Subir Dataset XML*: permite transmitir al servidor un archivo XML válido que especifica un conjunto de datos. Se envía el archivo como datos adjuntos de un mensaje SOAP. Los datos serán registrados en el servidor y seleccionados por defecto.
6) *Subir Función Transferencia JPG*: permite transmitir al servidor un archivo comprimido JPG que contiene el mapeo visual de un dataset previamente registrado. El archivo se envía como datos adjuntos de un mensaje SOAP. Los datos serán registrados en el servidor y seleccionados por defecto.

Los casos de uso previstos para la visualización de volúmenes se muestran en la Figura 5 y se describen a continuación:

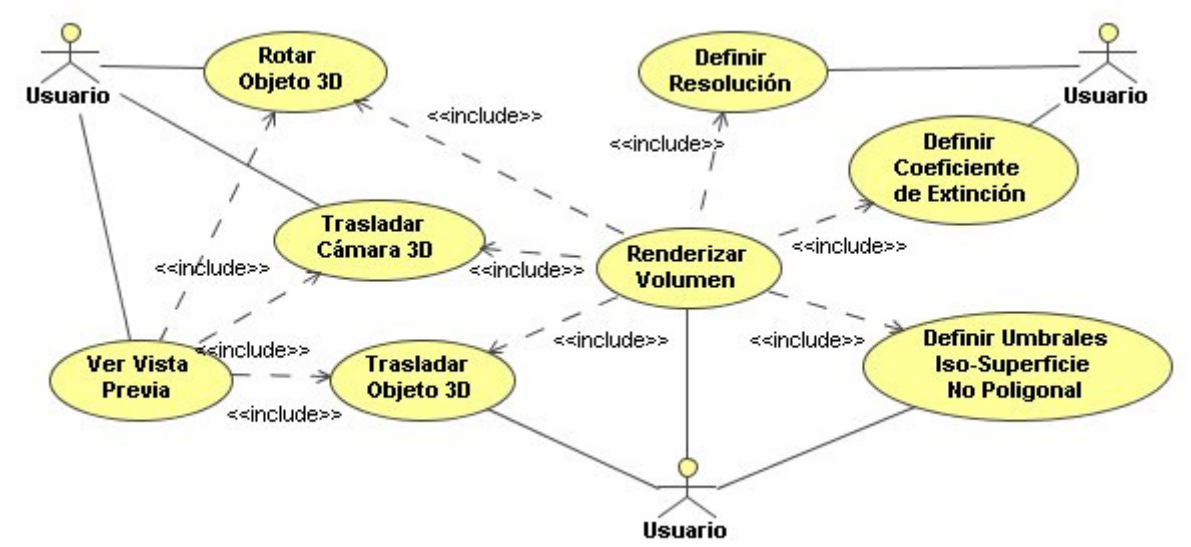

Figura 5: Diagrama de Casos de Uso - Interacciones del Usuario con el Servicio de Rendering

1) *Ver Vista Previa*: visualiza la superficie poligonal del volumen (también llamada objeto) con posición y orientación por defecto. La superficie es obtenida invocando al servicio *Vista Previa*, el cual retorna un archivo XML conteniendo la malla indexada de triángulos. Inicialmente el objeto debe estar centrado en el origen y contenido dentro de la pirámide de visión. La cámara estará alejada *Z* unidades del origen.
2) *Rotar Objeto 3D*: permite al usuario establecer la orientación del objeto mediante movimientos del ratón a través de un ArcBall [20]. Asumiendo que el objeto está contenido dentro de una esfera, el usuario puede definir interactivamente un arco en la superficie de la misma, haciendo que el objeto contenido rote cierto ángulo proporcional a la longitud del arco. El arco puede ser mostrado como guía adicional para el usuario.
3) *Trasladar Cámara 3D*: permite trasladar espacialmente el punto de visión. El punto de visión sólo se moverá en un plano paralelo al plano de visión el cual es por defecto el *XY*. Esta restricción permite conservar la coherencia con la rotación definida por el ArcBall, de otro modo el movimiento del punto de visión introduciría una nueva rotación en la escena.
4) *Trasladar Objeto 3D*: permite trasladar el objeto. La traslación se restringirá al eje ortogonal al plano de visión, por defecto el eje *Z*, siendo esta traslación la que permite el zoom geométrico.
5) *Definir Resolución*: permite definir la resolución final que tendrá el Rendering de Volúmenes.

6) *Definir Coeficiente de Extinción*: permite definir un coeficiente de extinción de luz ad hoc.
7) *Definir Umbrales Iso-Superficie No Poligonal*: permite definir el intervalo de valores del dominio que será visualizado. Permite definir un intervalo normalizado de los valores del dominio que se visualizarán, e iso-superficies no poligonales.
8) *Renderizar Volumen*: permite visualizar tanto el interior como la superficie del volumen seleccionado aplicando el mapeo visual de la Función de Transferencia seleccionada. Tomando los parámetros de visión definidos para la vista previa, se invocará al servicio *Volume Rendering* el cual retornará una imagen comprimida en formato JPG. La imagen será mostrada en una ventana emergente secundaria.

3.3 Implementación del Servicio de Datos

La solución propuesta acepta datasets que correspondan a una parametrización lineal sobre un dominio tridimensional discreto, tal que el dominio puede ser interpolado linealmente por piezas. Resulta práctico restringir los datasets a triangulaciones de *simplicials* sobre dominios discretos. De este modo, es posible definir un dataset como una malla indexada de tetraedros, sin perder generalidad. Cada vértice de la malla posee una magnitud física del dominio, que se guarda como un escalar, y cada tetraedro comparte sus vértices con otros tetraedros.

El *Servicio de Datos* que se definió conceptualmente es simplemente una abstracción que engloba tres servicios web reales: i) *File Upload*, ii) *Registrar y Buscar Dataset*, y iii) *Compilar Dataset.* El servicio *File Upload* gestiona el flujo de datos por la red. El servicio *Registrar y Buscar Dataset* registra la ubicación y el estado de cada dataset y Función de Transferencia en la base de datos y gestiona en el servidor correspondiente los datos. El servicio *Compilar Dataset* transforma los datasets del formato XML original a un formato especial como se explicará más adelante.

Debido a que no existen formatos estándar para representar las mallas de tetraedros, existen dos posibles enfoques de solución: i) el servicio *Compilar Dataset* procesa todos los formatos conocidos, o ii) se define un formato particular usando eXtensible Markup Language (XML). Una de las ventajas de usar XML es la posibilidad de utilizar XML-Schema - un meta lenguaje estándar que permite definir el formato de archivos XML. La solución propuesta opta por la segunda alternativa - utilizar datasets definidos en formato XML y validados en base al XML-Schema. La decisión toma en cuenta fundamentalmente que XML es actualmente un estándar ampliamente adoptado, si bien los archivos XML no resultan los más adecuados para almacenar datos numéricos voluminosos- son archivos de texto, donde cada caracter puede ocupar de uno a cuatro bytes y poseen gran número de etiquetas. La arquitectura de servicios web propuesta cambia eficiencia de almacenamiento y velocidad de transmisión por generalidad y facilidad de mantenimiento.

Los datasets XML y Funciones de Transferencia .JPG viajan por la red como archivos adjuntos tipo DIME, es decir, adjuntos a los mensajes SOAP. El servicio *File Upload* recibe los datos adjuntos y los almacena en el servidor. En la implementación hemos usado Web Services Enhancements (WSE) y en particular WS-Attachment para la transmisión de archivos grandes, fundamentalmente por su eficiencia, teniendo la posibilidad de transmitirlos por trozos (chunks) que fácilmente caben dentro de la memoria y viajan más rápido por la red. Debido al gran volumen de estos archivos resultaría imposible enviarlos dentro del cuerpo del envelope de SOAP.

El servicio *File Upload* se encarga de recibir y almacenar los datasets XML; inmediatamente el servicio *Compilar Dataset* compila los datasets XML en dos etapas. Primero parsea el código XML verificando que sea bien formado y válido, luego verifica que los datos correspondan a una malla

indexada de tetraedros con campo escalar. Finalmente, crea un archivo binario el cual será consumido por los Servicios de Rendering. En nuestra implementación el análisis y validación del código XML se hace mediante las clases *XmlTextReader* y *XmlValidatingReader* del Framework .NET las cuales parsean y validan el archivo XML. La razón de compilar los datasets XML es, principalmente, la eficiencia del sistema. El servicio *Compilar Dataset* se encarga de compilar y procesar los datos XML una única vez y dejarlos listos para que otros servicios los utilicen.

El servicio *Registrar y Buscar Dataset* se encarga de registrar el nombre y la ubicación de los datasets compilados en una base de datos. Los datasets compilados pueden estar guardados en distintas carpetas del disco, en servidores remotos o inclusive en la Web. A cada dataset se le asigna un identificador único para ser manejado internamente por todos los servicios web; adicionalmente se guarda el estado del dataset y una descripción útil para los usuarios. De este modo los datasets quedan distribuidos por la Web, sin embargo el acceso a la base de datos está centralizado en un solo servidor de bases de datos que, en nuestro caso, no es público.

Ningún servicio web, con excepción de los servicios *File Upload* y *Registrar y Buscar Dataset,* tiene acceso a la base de datos. Los servicios *Vista Previa* y *Volume Rendering* invocan al servicio *Registrar y Buscar Dataset* sólo para consultar la base de datos y obtener la ubicación de un dataset o Función de Transferencia. Si los datos residen en la Web, la transmisión directa es necesaria. Si un dataset cambia de nombre o de ubicación física, la base de datos quedaría apuntando a un recurso nulo. Es deseable que algún proceso se encargue de eliminar estos registros inválidos después de confirmar que el recurso al que hace referencia no existe. Debido que tanto las aplicaciones cliente como los servicios web hacen uso de los servicios *File Upload* y *Registrar y Buscar Dataset* debemos usar un gestor de bases de datos de acceso compartido. En nuestra implementación utilizamos un driver ODBC de Access que admite acceso compartido. Para nuestro propósito requerimos de una base de datos extremadamente simple, que consta de dos tablas fácilmente gestionadas por las clases del Framework .NET. Estas clases leen y escriben datos rápidamente.

3.3 Implementación del Servicio de Rendering

El *Servicio de Rendering* es la abstracción que incluye dos servicios web reales: *Vista Previa* y *Volume Rendering*. El servicio *Vista Previa* calcula sólo la superficie poligonal del volumen mientras que el servicio *Volume Rendering* genera la representación verdadera del interior y el exterior del volumen según el mapeo visual seleccionado.

Los Servicios de Rendering deben tener muy buena performance ya que hacen uso intensivo de los recursos del sistema. Es posible que estos servicios hagan uso de componentes o librerías escritas en lenguajes nativos como C o C++. En este caso, se debe sincronizar el acceso tanto a los objetos COM como DLL para que sean usados por varios clientes, ya que éstos suelen ser estáticos o globales. En caso que dos clientes intenten usar al mismo tiempo el servicio *Volume Rendering*, el sistema debe poner a uno de ellos en espera; esto obviamente crea contienda. En el caso del servicio *Volume Rendering* la contienda es inevitable ya que no es óptimo compartir la placa de video con N clientes. Por otro lado, si se dispone de varias placas de video o varios procesadores se podría reducir drásticamente la contienda creando un thread para cada cliente. Para sincronizar el acceso a recursos únicos usando ASP.NET hacemos uso de las primitivas *Lock* y *Unlock* que son justamente las que crean una zona de exclusión mutua en la cual un solo cliente accede al procesador en un instante dado.

En nuestra implementación el servicio *Volume Rendering* utiliza una librería propia escrita en C++ que implementa la Proyección de Tetraedros de Shirley y Tuchman [11] posteriormente mejorada por Stein *et al* [12]. La librería ejecuta 100% en el procesador principal (no usa aceleración por hardware) y esta compilada en una DLL para ser exportada al framework .NET. Notamos que desde un servicio web de ASP.NET no tenemos acceso al sistema de ventanas, esto hace imposible iniciar adecuadamente DirectX y OpenGL. Se espera que en futuras versiones dichas APIs ofrezcan *off-screen rendering*. Por ahora solo es posible hacer off-screen rendering con OpenGL usando extensiones específicas de algunas placas de video.

Los algoritmos convencionales de rendering de volúmenes tales como Proyección de Tetraedros y Ray Casting requieren calcular normales, interpoladores (ya sean gradientes o coordenadas baricéntricas) y tablas de adyacencia para cada dataset. Consideramos óptimo hacer estos cálculos una sola vez y almacenarlos en archivos binarios que puedan ser consumidos rápidamente múltiples veces. Dichos archivos se guardan en la misma ubicación y con el mismo nombre que el dataset pero con distintas extensiones. Como consecuencia, se invierte el orden en que se deben dar los pasos en el Modelo Unificado de Visualización [9]. La razón se debe a que no se esta diseñando un sistema de visualización completo sino solo el rendering. En el contexto de un sistema de visualización, el precalculo de datos estaría excluido del servicio *Volume Rendering*.

El servicio *Volume Rendering* potencialmente procesará datasets distintos por cada cliente. En este caso, debe descargar de memoria el dataset usado por el cliente anterior y cargar el dataset del cliente actual. Sin embargo, si varios clientes desean visualizar el mismo dataset, a fin de optimizar performance el servicio no debería descargar los datos de memoria. Si dos procesos usan consecutivamente el mismo dataset, nos aseguramos que el dataset no sea descargado de memoria simplemente revisando el ID del dataset que se desea visualizar - si es igual al ID actualmente cargado en memoria, éste se mantiene; en caso contrario se carga el nuevo dataset. Esto se puede lograr guardando el ID en la memoria compartida que provee el objeto *Application* de ASP.NET.

El servicio *Vista Previa* provee al usuario de una metáfora visual que lo ayuda a definir *cómo* quiere la visualización. Concretamente el usuario debe definir parámetros de la vista; por ejemplo, el ángulo y posición espacial del volumen entre otros. *Vista Previa* se encarga de calcular la superficie del volumen y simplificarla geométricamente con el fin de generar una representación muy liviana del dataset que pueda ser manipulada interactivamente. El resultado es un archivo XML que contiene una malla indexada de triángulos que se envía al cliente mediante WS-Attachments. Usamos XML - del mismo modo que con las mallas de tetraedros, a falta de un estándar para representar mallas indexadas de triángulos. En nuestra implementación el servicio *Vista Previa* genera rápidamente un modelo aproximado y de pocos polígonos del volumen calculando el convex hull de los vértices que lo conforman según el algoritmo Quick Hull [2]. El convex hull nos sugiere la silueta del dataset. Desafortunadamente el convex hull no es una alusión precisa de la forma real del volumen; se requiere de un modelo aproximado mas exacto que se calcula usando técnicas de simplificación.

La aplicación cliente se encarga de visualizar la malla de triángulos y de proveer una interfase mediante la cual el usuario pueda, interactivamente, definir parámetros de visión que se enviarán al servicio *Volume Rendering*. En nuestra implementación la aplicación cliente provee una interfase sencilla e intuitiva basada en las rotaciones mapeadas en la esfera [20], acercamientos y movimientos ortogonales de la cámara. Notamos que combinando estas sencillas transformaciones es posible definir cualquier vista del volumen.

El trabajo del cliente no es ser un simple *front end* que invoca a los Servicios Web. Es necesario que el cliente recopile de forma visual e interactiva los parámetros que se usaran en la invocación del servicio *Volume Rendering*. Cuando el usuario hace una pre-visualización del dataset actual, se visualiza la geometría devuelta por el servicio *Vista Previa* con los parámetros iniciales por defecto –la matriz de rotación es la identidad, matriz de visión esta alejada Z unidades del origen, matriz de proyección por defecto con 60 grados de amplitud. Después el usuario puede interactuar con la geometría de *Vista Previa*, modificando en tiempo real las matrices y parámetros de visión usando una interfase de alto nivel. Finalmente tomando las matrices y parámetros calculados durante la vista previa o los parámetros por defecto, podrá invocar al servicio *Volume Rendering* para recibir una imagen real del interior y exterior del volumen.

Debido a que la geometría de vista previa es siempre la misma, solo es necesario pedirla una sola vez al servicio *Vista Previa* y se guarda en el disco local para obtenerla rápidamente a posteriori. De la misma manera el servicio *Vista Previa* sólo calcula la geometría simplificada una sola vez por cada dataset, luego solo busca una copia en el disco local y la transmite mediante WS-Attachments.

La Figura 6 muestra el rendering de volúmenes del dataset SPX (ventana pequeña) con el ángulo y posición especificados en la vista previa del dataset que se ve en amarillo.

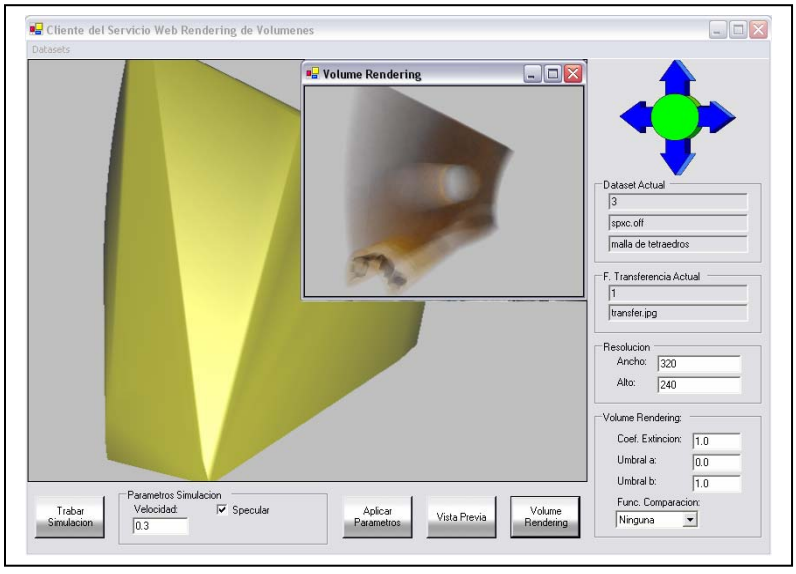

Figura 6: Rendering de Volumen

4 CONCLUSIONES

El presente trabajo presenta una arquitectura orientada a servicios para rendering de volúmenes no estructurados. Se presentó la especificación funcional de la aplicación mediante casos de uso y detalles de implementación. Se definió una interfase de alto nivel para el usuario, en base a rotaciones mapeadas en la esfera y traslación ortogonal de la cámara. La implementación utilizó la tecnología ASP.NET, comunicándose con los servicios web mediante RPC y SOAP.

En la implementación se utilizó DIME y WS-Attachments que permitió transmitir archivos de gran tamaño e inclusive dividirlos en trozos (chunks). Las transmisiones del servicio *Volume Rendering* fueron exitosas. Este servicio retorna el resultado de la visualización como un archivo mapa de bits comprimido JPG que ocupa solo unos cientos de kilobytes (alrededor de 300KB a resolución

1024x1024). Así mismo, el servicio *Vista Previa* retorna geometría simplificada que ocupa unos cientos de kilobytes, y resulta adecuado para un attachment.

Los servicios de rendering requieren que los datasets físicamente distribuidos en la red sean potencialmente transmitidos cada vez que son invocados. Si bien la transmisión de datasets puede reducirse implementando un sistema de caché local simple, el cual evitaría la retransmisión de los datasets usados constantemente, esto no es suficiente ante un numero creciente de peticiones ya que la caché colapsa y se hace necesaria la retransmisión de datos. Por esto, es deseable que los datasets residan donde se los requiere frecuentemente y que sean eliminados a petición del cliente o por expiración de días.

Debido a la carencia de *off-screen rendering* por parte de las APIs DirectX y OpenGL en Windows, no hacemos uso de la placa de video para los procesos de rendering en los servicios web, por el contrario usamos una implementación propia de proyección de tetraedros compilada en una DLL que ejecuta 100% en el procesador del sistema.

BIBLIOGRAFIA

[1] Ang C. S., Martin D. C., Doyle M. D.: *"Integrated Control of Distributed Volume Visualization Through the World Wide Web"*. In Proceedings of IEEE Visualization '94 (1994), IEEE Computer Society Press. 22

[2] C. Bradford Barber, David P. Dobkin, Hannu Huhdanpaa, *"The Quick Hull Algorithm for Convex Hulls"*, January 9, 1995

[3] CurVis: http://www.icg.informatik.uni-rostock.de/Projekte/MoVi/mirror/proto.html

[4] K.W. Brodlie, D.A. Duce, J.R. Gallop, J.P.R.B. Walton, J.D. Wood, *"Distributed and Collaborative Visualization"*, The Eurographics 2004

[5] Stuart M. Charters, Nicolas S. Holliman, Malcolm Munro.: *"Visualisation on the Grid: A Web Service Approach"*, 2005

[6] Engel K., Hastreiter P., Tomandi B., Eberhardt K., Ertl T.: *"Medical Volume Rendering over the WWW"*. In Proceedings of IEEE Visualization 2000 (2000), IEEE Computer Society, pp. 449–452. 21

[7] Engel K., Ertl T.: *"Texture-based Volume Visualization for Multiple Users on the World Wide Web"*. Proceedings of the Eurographics Workshop in Vienna, Austria (1999), pp. 115–124. 21

[8] Ian J. Grimstead, Nick J. Avis, David W. Walker, Roger N. Philp, *"Resource-Aware Visualization Using Web Services"*, 2004

[9] Martig S, Castro S, Fillottrani P, Estévez E.: *"Un Modelo Unificado de Visualización"*, Proceedings 8vo.Congreso Argentino de Ciencias de la Computación. La Plata, Argentina. Octubre, 2003.

[10] McCormick B., Defanti T, Brown M.: *"Visualization in Scientific Computing"*. Computer Graphics 21, 6 (1987).

[11] Shirley P., Tuchman A., *"A Polyhedral Approximation to Direct Scalar Volume Rendering"*, Computer Graphics 24 (November, 1990), 63–70.

[12] Stein C, Becker B, Max N, *"Sorting and hardware assisted rendering for volume visualization"*, In Proceedings of the 1994 Symposium on Volume Visualization, pages 83-89, October 1994.

[13] Yunsong Wang, *"Visualization Web Service"*, Master's Thesis, Florida State University, 2003.

[14] Yeo A.: *"Client-based Web Visualization"*. MSc.thesis.School of Computer Studies, University of Leeds, U.K., 1998.

[15] World Wide Web Consortium, "*Web Services Glossary*", http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

[16] Especificación de VRML: http://www.web3d.org/x3d/specifications/vrml/

[17] IRIS Explorer: http://www.nag.co.uk/welcome_iec.asp

[18] IBM Open Visualization Data Explorer: http://www.research.ibm.com/dx/

[19] Visible Human Server: http://visiblehuman.epfl.ch/

[20] Shoemake, K.: *"ARCBALL: A user interface for specifying three-dimensional orientation using a mouse"*, proceedings of Graphics Interface (1992) 151-156.´

[21] World Wide Web Consortium, Web Services Description Language – WSDL, http://www.w3.org/TR/wsdl

[22] World Wide Web Consortium, Simple Object Access Protocol - SOAP, http://www.w3.org/TR/soap/

[23] Erl, Thomas. Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall, 2005.

[24] OASIS – UDDI, Universal Description, Discovery and Integration, http://www.uddi.org/