Contenedor de Aplicaciones para Computación Gráfica

César Julio Bustacara Medina, Oscar Javier Chavarro, David Arturo Díaz Díaz

Pontificia Universidad Javeriana, Dept. Ingeniería de Sistemas
Bogotá D.C., Colombia
{cbustaca, ochavarr, d-diaz}@javeriana.edu.co

Resumen. La arquitectura del contenedor de aplicaciones para computación gráfica surgió como un modelo poderoso de reutilización, en el cual se hace distinción entre la funcionalidad de una aplicación y el conjunto de servicios que le dan soporte [1]. Las aplicaciones de computación gráfica comparten un conjunto de servicios como el render o la simulación que pueden ser separados de la funcionalidad puntual de las mismas. Este artículo presenta la arquitectura de un contenedor que permite alcanzar la reutilización de los servicios comunes de las aplicaciones de computación grafica y facilita el proceso de implementación y trazabilidad.

Keywords: Contenedor de Aplicaciones, Arquitectura de Software, Computación Gráfica.

1 Introducción

Para alcanzar el éxito en el desarrollo de aplicaciones para cualquier familia de sistemas, el paso primordial es la definición de los requerimientos, los cuales se dividen en funcionales y no funcionales. Los requerimientos funcionales son únicos para cada aplicación y comprenden el conjunto de reglas del negocio establecidas que satisfacen las necesidades del usuario y, que, a través de operaciones bien definidas, le ayudan a desempeñar sus funciones [1]. Por otro lado, los requerimientos no funcionales son generales y pueden ser compartidos por un conjunto de aplicaciones [2].

Específicamente, en el desarrollo de aplicaciones para computación gráfica (CG) se ha detectado una serie de requerimientos funcionales que están presentes en la mayoría de sistemas, lo cual implica que se pueden convertir en requerimientos no-funcionales. Dentro de los requerimientos funcionales están: la construcción de representaciones geométricas; la evolución temporal de las escenas (simulación); la habilidad de consultar las escenas o modelos; manejo fácil y eficiente de dispositivos I/O especializados; construcción de escenarios controlados; soporte para la construcción de aplicaciones de CG, realidad virtual y realidad aumentada; capacidad de modificar las aplicaciones, escenarios, comportamientos y geometrías de forma interactiva. Por otra parte, los

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 843-852, 2008.

requerimientos no funcionales incluyen: la portabilidad, escalabilidad, modularidad, eficiencia y aprovechamiento de recursos distribuidos y de dispositivos especializados.

Actualmente existen varias arquitecturas de software para el desarrollo de aplicaciones en computación gráfica como Maverik [3] [4], VTK [5], MRToolkit [6], VRJuggler [7], NetJuggler [8], Avocado [9] y AQUYNZA [10]; plataformas de juegos como OGRE, Crystal y VRTools. Pero ninguna de ellas logra satisfacer el conjunto completo de requerimientos funcionales y no-funcionales necesarios para construir aplicaciones de computación gráfica de una manera rápida y eficiente.

El primer aspecto en común que se encuentra en la mayoría de estas arquitecturas es el soporte para el manejo de dispositivos de interacción hombre-máquina, como guantes de datos y cascos de realidad virtual (HMD). Usualmente, las arquitecturas también posibilitan y facilitan técnicas de multiproyección, estereopsis y realidad virtual cooperativa. Sin embargo, algunas de las arquitecturas actuales, a pesar de soportar un amplio rango de requerimientos, todavía se basan en el antiguo paradigma de la programación estructurada, ofreciendo APIs de control basadas en un gran número de funciones y procedimientos, dificultando su mantenimiento, modificación e integración.

Los entornos virtuales en computación gráfica tienden a ser cada vez más complejos, voluminosos y exigentes computacionalmente. Una estrategia fundamental para poder lograr resultados prácticos con calidades razonables de visualización en éste escenario, es el aprovechamiento de la programación en paralelo, en agrupaciones (clusters) y grillas (grid computing). De las arquitecturas de CG revisadas únicamente Avocado tiene en cuenta éste aspecto fundamental.

Por las razones expuestas antes, en este artículo se describe la arquitectura de un contenedor específico para aplicaciones de CG, la cual permite superar algunas de las desventajas enunciadas. El contenedor proporciona herramientas para la representación geométrica, un esquema de desarrollo orientado por objetos, aprovecha la programación paralela y soporta un grafo de escena propio y configurable que permite incluir algoritmos avanzados de recorte de escena. Pero un elemento crucial y que impacta sobre los requerimientos no-funcionales, es que el contenedor de aplicaciones para CG permite soportar gran parte de los atributos de calidad deseados por cualquier aplicación [1]. El contenedor ofrece un conjunto de servicios que promueven la reutilización y la simplificación en la construcción de aplicaciones, ya que los servicios son prestados por medio de un corredor de servicios.

2 Antecedentes

Dos de los problemas más importantes que existen en la computación gráfica desde su concepción son: la complejidad que tienen las aplicaciones en su fase de construcción y la alta demanda de procesamiento que requieren. Como elemento teórico se consideran arquitecturas importantes en el ámbito de la computación gráfica como: VR Juggler, Avocado, OpenGL Performer, OpenInventor, X3D (Extensible 3D).

VRJuggler: En general, VRJuggler tiene como objetivo principal reducir la complejidad. Para esto provee una plataforma virtual, que es una capa entre el sistema operativo y la aplicación específica, que se encarga de resolver por medio del encapsulamiento y la abstracción los siguientes problemas [7]: generar una abstracción de la complejidad del desarrollo de sistemas de realidad virtual; dar transparencia en la extensibilidad del sistema por medio de software; dar flexibilidad en cuanto al soporte de hardware; realizar el cambio dinámico de aplicaciones por medio de la reconfiguración de módulos; soportar el procesamiento en paralelo; y generar métricas en tiempo real sobre el rendimiento del sistema[7] [11].

Avocado: Es un framework para aplicaciones de entornos virtuales distribuidos en el cual el servidor es el elemento central de la plataforma para todos los componentes relacionados. Posee un modelo de eventos que permite la interacción entre todos los componentes. Los nodos proporcionan un grafo de escena orientado por objetos que es usado por la API para representar y renderizar geometrías complejas. Todos los componentes son contenedores que representan el estado de un componente como una colección de campos [9].

OpenGL Performer: es una potente y completa interfaz de programación para los desarrolladores, la cual facilita la creación de aplicaciones de simulación en tiempo real y otras que requieran alto desempeño. Simplifica el desarrollo de aplicaciones complejas para simulación, fabricación, diseño, realidad virtual, visualización científica, entretenimiento interactivo, multidifusión de vídeo, paseos arquitectónicos y el diseño asistido por computador. OpenGL Performer permite lograr máximo rendimiento y hacer un uso óptimo del sistema, adicionalmente, permite manejar sofisticadas características para gráficos en 3D. Ofrece posibilidad para escalar fácilmente el procesamiento a múltiples procesadores gráficos y múltiples arquitecturas de tuberías, desplegar las aplicaciones en una amplia gama de sistemas, y adaptarse a múltiples sistemas de gráficos de hoy y del futuro [13].

Open Inventor: de la compañía TGS Visual Concepts es la herramienta más utilizada por los desarrolladores en C++ y Java para desarrollar aplicaciones de graficas 3D multiplataforma y orientadas por objetos. Satisface las necesidades de desarrollo rápido de aplicaciones, visualización 3D en tiempo real, incremento de la productividad y la optimización de los costos de desarrollo y recursos. Open Inventor ofrece una serie características importantes para crear, actualizar y manipular gráficas 3D, tales como: volume rendering, manejo de grandes conjuntos de datos, interacción en tiempo real y software al lado del servidor que genera imágenes en 3D a través de la red. Open Inventor permite desarrollar en C++ o Java, y también proporciona la flexibilidad de funcionamiento de las aplicaciones desarrolladas en diferentes plataformas [12].

X3D (Extensible 3D): es la próxima generación del estándar abierto para la Web. Es el resultado de varios años de desarrollo por parte del Grupo de Trabajo X3D del Consorcio Web 3D y el reciente Grupo de Trabajo Browser. Este último ha trabajado de cerca con el Grupo de Trabajo X3D para crear una nueva especificación X3D que ofrezca compatibilidad con el existente contenido VRML, browsers y herramientas.

Cada una de las arquitecturas existentes para CG posee ciertas características a nivel de requerimientos tanto funcionales como no funcionales que satisface. En la tabla 1 se muestra un resumen de algunas arquitecturas y características importantes, las cuales permiten identificar la existencia de algunas limitaciones, por ejemplo, debilidades en cuanto al manejo de dispositivos de entrada y salida, procesamiento distribuido, lenguajes de desarrollo, persistencia y contenedores de aplicaciones.

Tabla 1. Comparación de requerimientos de arquitecturas para CG [9]

Característica	OpenGL Performer	OpenInventor Extensible 3D	(X3D)	VR Juggler
Sistema Operativo	SGI IRIX, I386 Linux	SGI IRIX, I386 Linux	N/A	SGI IRIX, I386 Linux, Microsoft Windows
Distribución	Producto comercial	Proyecto de código abierto	Especificación, implementación no disponible	Proyecto de código abierto
API gráfica de bajo nivel	OpenGL	OpenGL	N/A	Especifica de la aplicación
Estructura del sistema	Toolkit	Framework	N/A	Framework
Modelo de ejecución	Multiples hilos	Un solo hilo	N/A	Múltiples hilos
Escalabilidad	Múltiples procesadores, Múltiples subsistemas gráficos	Ninguno	N/A	Dependiente de la aplicación
Opciones visualización	Mono y/o estéreo, múltiple	Mono o estéreo	N/A	Mono y/o estéreo, múltiple, dependiente de la aplicación
Modelo de objetos	Grafo de escena con las transformaciones anidadas	Grafo de escena con las transformaciones anidadas	Grafo de escena con las transformaciones anidadas	Ninguno, dependiente de la aplicación
Persistencia	Persistencia del grafo de escena, Formato binario de Performer (.pfb)	Persistencia del grafo de escena, Formato OpenInventor, Texto o Binario (.iv)	Persistencia del grafo de escena, VRML97, XML y Archivos binarios	Ninguno, dependiente de la aplicación
Formatos de datos	Cargadores para todos los formatos de archivos 3D populares	Formato de OpenInventor, Texto o Binario (.iv)	Formato de archivo VRML, Texto o Binario (.iv)	Ninguno, dependiente de la aplicación
Dispositivos de entrada	Ratón, Teclado	Ratón, Teclado	N/A	Ratón, Teclado, Dispositivos de entrada más populares para RV
Mecanismos de extensión	Herencia, Composición	Herencia, Composición, Carga dinámica de clases	Herencia y Composición de objetos nativos, código script	Herencia, Composición
Lenguaje de desarrollo	C, C++	C++	Disponibilidad de IDL para C++, Java, JavaScript	C++
Contenedor de aplicaciones	N/A	N/A	N/A	N/A

3 Arquitectura del Contenedor

Para adoptar la arquitectura de un contenedor, un conjunto de servicios comunes a los componentes potencialmente desplegables debe ser encontrado. La arquitectura JEE define un conjunto de servicios necesarios para las aplicaciones empresariales, entre los cuales está la persistencia, la seguridad y el manejo del ciclo de vida para los

componentes. El objetivo de JEE es evitar la implementación de dichos servicios al incluirlos en la aplicación de manera declarativa [7] [14].

Con las aplicaciones de CG, un conjunto de necesidades comunes en las aplicaciones fue encontrado y su descripción fue detallada. Estas necesidades son consideradas como servicios para las mismas. Un servicio encapsula la implementación de procesos necesarios para dar cumplimiento a los requerimientos funcionales y no-funcionales en una aplicación.

Un objetivo del contenedor es promover la reutilización; para esto provee servicios que dan respuesta a los requerimientos. Por consiguiente, los servicios para aplicaciones en CG se convierten en requerimientos funcionales para el contenedor. Si por ejemplo, una aplicación necesita desplegar un conjunto de datos en pantalla, para dicha aplicación el render no es un requerimiento funcional, ya que se limita a presentar datos, y no define de manera directa el propósito de la aplicación. El render es un servicio que debe ser incluido en la aplicación de manera declarativa; entonces para el contenedor, el render es un requerimiento funcional. El requerimiento funcional de render, para el contenedor tiene requerimientos no-funcionales asociados, por ejemplo, flexibilidad para escoger distintos algoritmos de render en tiempo de ejecución. Siguiendo esta filosofía deductiva fue posible identificar los requerimientos generales del contenedor [14].

3.1 Características del Contenedor

La CG es candidata a ser adaptada a la arquitectura de un contenedor, ya que de ésta puede derivarse un conjunto de servicios recurrentes a las aplicaciones de este tipo de familia. El conjunto de servicios a prestar reside en un contenedor y han de ser prestados a componentes que cumplan con la especificación. Entonces, los componentes se limitan a implementar los requerimientos funcionales de la aplicación y declarar que servicios necesitan del contenedor. Algunas de las ventajas que trae la arquitectura de un contenedor a la computación gráfica se enuncian a continuación: facilidad de uso, prototipado rápido, implementación diversa de servicios, reutilización, configuración del sistema, trazabilidad en el proceso de desarrollo de software, aproximación modular, extensibilidad, flexibilidad y composición arquitectónica.

3.2 Descripción

El contenedor es un programa para desplegar aplicaciones que han sido realizadas siguiendo el marco de trabajo VSDK (Vitral Software Development Kit); ofrece un conjunto de servicios y es su responsabilidad proveer la implementación de los mismos. Una aplicación implementa su funcionalidad principal y declara el conjunto de servicios que necesita obtener del contenedor.

El contenedor provee la implementación de los servicios declarados por la aplicación [2]. Cabe aclarar que pueden existir varias implementaciones del mismo servicio. El

cambio del componente que implementa la interfaz cambia el comportamiento de una aplicación. Por ejemplo, el servicio de render tiene una interfaz única pero puede ser implementado con ray tracing [8] o Z-Buffer [9], razón por la cual, la configuración de los servicios debe poder hacerse a nivel de aplicación o de contenedor. En el momento de hacer el despliegue de la aplicación dentro del contenedor, los servicios no implementados son asumidos por el contenedor.

El contenedor de aplicaciones para el VSDK da cumplimiento a un conjunto definido de servicios que debe ser extensible y además flexible; debe cumplir con los requerimientos funcionales y no funcionales encontrados. Para dar cumplimiento a los servicios propuestos se usa una arquitectura en capas (layers) [10], donde cada capa cumple con las siguientes responsabilidades:

Capa supportPlataform: Es la capa que soporta las operaciones básicas necesarias para construir los servicios del toolkit y la capa de integración.

Capa toolkitVSDK & integration: Es la capa que da soporte a los servicios de alto nivel descubiertos tras el análisis de los requerimientos funcionales y no funcionales.

- *toolkitVSDK*: Es un conjunto de herramientas necesarias para implementar los servicios de alto nivel que va a prestar el marco de trabajo a los desarrolladores. El toolkit está dividido en paquetes que agrupan clases con funcionalidad similar. Entre otros componentes gráficos, cuenta con mallas de alambre, cámaras, controladores de cámara, componentes de presentación y manejo de dispositivos de salida.
- *integration*: Ofrece un conjunto de servicios que comunican un contenedor VSDK con sistemas externos como bases de datos, sistemas legacy, entre otros. Está diseñado para tener extensibilidad a futuro. Éste es uno de los tres módulos que comprenden el kernel del contenedor VSDK.

Capa ServiceManagers y Communication

- ServiceManagers: Los manejadores de servicio son los módulos encargados de proveer los servicios de CG necesarios en las aplicaciones. Es importante dejar la puerta abierta a la definición y creación de nuevos módulos para expandir la funcionalidad del contenedor.
- Communication: El modulo de comunicación ofrece el conjunto de servicios necesarios establecer la comunicación entre dos o más contenedores.

Capa ServiceBroker: Esta capa oculta la complejidad tanto de número de servicios como de la variedad de formas de prestarlos, obteniendo así un conjunto ordenado y accesible de servicios; su objetivo es hacer sencilla la construcción de aplicaciones. Para acceder a los servicios, la aplicación obtiene referencias a componentes por medio de la interfaz del ServiceBroker.

Capa Application: Es un conjunto de componentes creados por un desarrollador. Dichos componentes exponen la funcionalidad de la aplicación, delegando al contenedor servicios como el render, la comunicación y la integración. Desde el punto de vista del contenedor, el conjunto de requerimientos funcionales de una aplicación define servicios que la aplicación tiene que prestar y además, que la hacen distinta a las demás aplicaciones. Bajo

éste orden de ideas, el render es un servicio a ser prestado por el contenedor ya que muchas aplicaciones pueden reutilizar el mismo algoritmo de render.

4 Extensibilidad de la Arquitectura por Medio de la Comunicación y la Integración

El contenedor cuenta con servicios de comunicación e integración, lo que permite generar proyectos con arquitecturas distribuidas, aprovechando las ventajas de las mismas.

El modulo de comunicación: tiene como fin dar respuesta a los requerimientos funcionales y no funcionales que de alguna manera enuncian comunicación entre contenedores; se descubrieron dos unidades cohesivas necesarias. La primera tiene por objeto dar respuesta a necesidades de distribución de tareas altamente paralelizables [11] y la segunda proveer facilidades para compartir datos entre varios contenedores, para lograr generar espacios virtuales compartidos [12] con información sincronizada [13].

- **Distribución de tareas altamente paralelizables**
  Una tarea puede ser dividida en partes siempre y cuando la ejecución concurrente de estas partes tenga un resultado igual a la ejecución secuencial de la tarea original [15]. La idea de ésta parte del módulo de comunicación es tomar una tarea y dividirla en partes que puedan ser procesadas en diferentes contenedores.
- **Distribución de una escena compartida por varios contenedores VSDK**
  Compartir información es la base para compartir escenarios virtuales. El contenedor VSDK tiene en su módulo de comunicación interfaces que proveen al desarrollador servicios para compartir datos entre contenedores. Lo importante es mantener un conjunto de datos actualizado entre los contenedores VSDK que soliciten tener información compartida.

El modulo de integración: uno de los servicios a prestar en el marco de trabajo es el de integración con sistemas heterogéneos para extender la funcionalidad del mismo, incorporando funcionalidad de sistemas desarrollados por terceros y ofreciendo también servicios a éstos. Sin embargo, la integración no es el objetivo principal del marco de trabajo, razón por la cual debe encontrarse un diseño extensible que permita la conexión con sistemas externos siendo al mismo tiempo sencillo y desarrollable. Una de las capas inferiores del contenedor es J2SE. Uno de los paquetes incluidos en ésta tecnología es JNDI [14], que provee una estructura sencilla y extensible que permite dar nombre a objetos para accederlos mediante éstos a posteriori; ésta facilidad permite definir un esquema de integración libre de implementación.

Se proponen algunos ejemplos de arquitectura que se basan en las facilidades ofrecidas por los módulos de comunicación e integración antes descritos.

Cliente/Servidor: La arquitectura cliente servidor tiene varias aplicaciones, por medio del módulo de comunicación, un contenedor VSDK puede tomar el rol del servidor y procesar solicitudes a contenedores VSDK que toman el rol de clientes. En la Figura 1, el contenedor del centro es el servidor y los de los extremos los clientes.

Modelo de 3 tiers: El modelo de 3 tiers separa al cliente, a la lógica del negocio y al recurso para evitar acoplamiento y así incrementar la flexibilidad. En la figura 2, el tier cliente tiene un contenedor que hace solicitudes por medio de su modulo de comunicación. El tier servidor tiene también un contenedor que procesa solicitudes y accede a la capa de recurso por medio del modulo de integración. Una vez procesada la solicitud, devuelve la respuesta por medio del modulo de comunicación. La capa de recurso tiene un recurso genérico y se conecta al modulo de integración del contenedor en el tier de lógica.

Procesamiento distribuido con varios nodos: Uno de los objetivos del marco de trabajo VSDK es incrementar el rendimiento de las aplicaciones en computación gráfica. Para esto, cuenta con facilidades de distribución en el modulo de comunicación. En la Figura 3 se muestra un proceso que distribuye la tarea de render en tres contenedores conectados a través de sus módulos de comunicación. El contenedor maestro orquesta la distribución y los contenedores esclavo llevan a cabo el procesamiento.

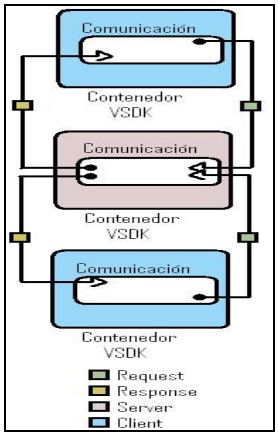

Figura 1. Cliente-Servidor

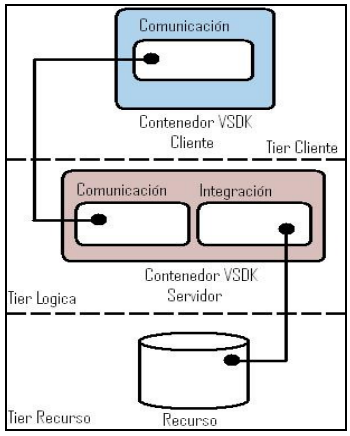

Figura 2. Arquitectura 3-Tier

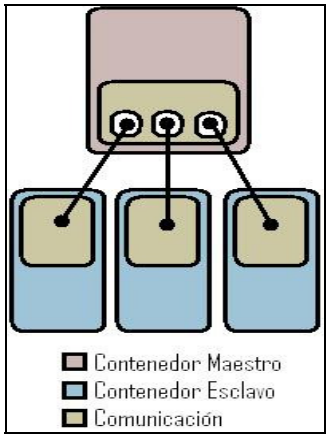

Figura 3. Distribución de procesos

5 Pruebas de Rendimiento para los Módulos Implementados

La Figura 4 muestra la prueba comparativa de rendimiento entre la implementación de render usando el contenedor y sin este. La razón por la cual el rendimiento es comparable es que el kernel se limita a brindar la implementación de un servicio; el control de dicho componente es manejado por el desarrollador de aplicaciones. La prueba contó con una escena en la cual la cantidad de polígonos va aumentando de manera controlada, y se observa el tiempo que le toma a la máquina dibujar dichos polígonos.

El rendimiento en la implementación actual del modulo de comunicación puede verse en la figura 5. La aplicación de prueba comparte datos entre varios clientes. A medida que aumenta la cantidad de clientes, también lo hace la cantidad de mensajes. El tiempo es aceptable solo hasta dos participantes dado que la aplicación de prueba envía mensajes en un protocolo orientado a conexión a cada uno; distintas implementaciones del módulo de comunicación haciendo uso de tecnologías broadcast podrían mejorar considerablemente este tiempo.

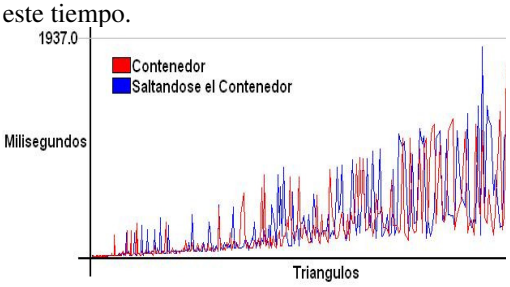

Figura 4. Rendimiento del Render

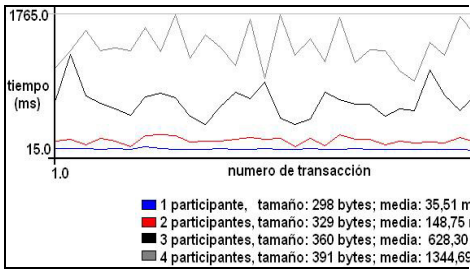

Figura 5. Rendimiento del modulo de Comunicación

6 Conclusiones

El contenedor VSDK funciona de manera correcta y por ser implementado usando el lenguaje Java posee características de portabilidad y soporte a múltiples plataformas, adicionalmente, provee una solución alterna a las existentes actualmente para los desarrolladores de aplicaciones para computación grafica.

Los desarrolladores en computación gráfica tendrán a su disposición un conjunto de componentes que implementan los servicios necesarios para desarrollar aplicaciones de manera ágil y ordenada. Dicho conjunto de componentes será un punto común para generar sinergia en el proceso de desarrollo. Basándose en el uso del mismo marco de trabajo, los desarrolladores podrán transmitir conocimiento de manera fácil, compartir sus ideas y entenderlas de otros, ya que se comunicarían en términos de los mismos componentes.

Debido a la disminución de la complejidad en la generación de aplicaciones que aprovechen las ventajas ofrecidas por la Computación Gráfica 3D, distintos sistemas podrán beneficiarse al presentar datos en ambientes 3D, incrementando la productividad de los usuarios y abriendo nuevas oportunidades de negocio.

La aproximación modular permite diseñar e implementar nuevos servicios y adicionarlos de manera transparente. La generación de módulos no es un trabajo difícil, de hecho, un desarrollador experimentado en computación gráfica puede desarrollar sus propios módulos y compartirlos con la comunidad que utiliza el marco de trabajo.

El acercar la computación gráfica a la arquitectura de software permite generar esquemas robustos que brindan mayor calidad de servicio a los usuarios finales. Las aplicaciones de computación gráfica a muchos niveles dependen ya de la distribución, prueba de ello son los sistemas de ubicación geográfica o los servicios de juegos multi-jugador a nivel global. El VSDK establece vínculos entre la computación gráfica y la distribución de componentes para generación de arquitecturas.

El marco de trabajo es una herramienta para desarrollar aplicaciones en computación gráfica, su propósito es hacer del desarrollo una disciplina ordenada y trazable, simplificando el proceso de ingeniería de software en dichas aplicaciones.

7 Bibliografía

1. D. R. Prescott. Functional requirements and attributes for records management in a component-based architecture. Technical report, Records Management Service Component, 2005.
2. B. Bruegge. *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*. Prentice Hall, 2000.
3. R. J. Hubbold. Maverik - the manchester virtual environment interface kernel. In *Eurographics workshop on Virtual environments and scientific visualization '96*, 1996.
4. R. Hubbold. Gnu/maverik: A microkernel for large-scale virtual environments. *Presence: Teleoperators and Virtual Environments*, 10(1):22-34, February 2001.
5. K. Inc. Vtk home page. http://www.vtk.org, May 2008.
6. C. Shaw. Decoupled simulation in virtual reality with the mr toolkit. *ACM Transactions on Information Systems (TOIS)*, 1993.
7. C. Just. Vrjuggler: A framework for virtual reality development. 2nd Immersive Projection Technology Workshop (IPT98), May 1998.
8. A. Bierbaum. Vr juggler: A virtual platform for virtual reality application development. In *Virtual Reality 2001 Conference (VR'01)*, 2001.
9. H. Tramberend. Avocado: A distributed virtual reality framework. In *IEEE Virtual Reality Conference 1999 (VR'99)*, 1999.
10. O. Chavarro. Aquynza: Plataforma para el desarrollo de aplicaciones de realidad virtual. Master's thesis, Universidad de los Andes, 2000.
11. A. Bierbaum. Vr juggler. http://www.vrjuggler.org/index.php, May 2008.
12. Open Inventor. Inc. Open inventor. http://oss.sgi.com/projects/inventor/, May 2008.
13. OpenGL_Performer. Opengl performer™ programming guides reference manuals. http://www.sgi.com/products/software/performer/overview.html, May 2008.
14. S. Gobel. *The COMQUAD Component Container Architecture*. Institute for System Architecture, Institute for Software Engineering, TU Dresden, Germany, 2005.