Arquitectura de un Contenedor de Aplicaciones para Computación Gráfica

Anamaria Ortiz[1], Diego González[1] y César Bustacara[1]

[1] Pontificia Universidad Javeriana, Dept. Ingeniería de Sistemas
Bogotá D.C., Colombia
{ anamaria.ortiz, de.gonzalez, cbustaca}@javeriana.edu.co

Abstract. The motivation of using an applications container is that it provides services to all applications that host. Adittionally, the majority of services are non-functional requirements, facilitating the applications development of both time and effort from developers. In computer graphics, an application container is a mechanism that allows the reusability, maintainability and extensibility of components, facilitates the applications deployment, reducing the complexity level for developers and allows decouple between logic (of applications) and different libraries available. Also, it separates the dependence between the programming language and graphics libraries (API's); additionally, the user does not need to worry about generic services that any computer graphics application requires. Also, it is platform independent, allowing that the developers are not bound to use a particular operating system, providing greater portability of developed components. This paper presents the applications container architecture, which allows achieving several non-functional and functional requirements of computer graphics applications.

Keywords: Software Architecture, Quality Attributes, Computer Graphics, Applications Container.

1 Introducción

El desarrollo de software desde sus inicios hasta hoy ha tenido una constante evolución, tanto a nivel tecnológico como de metodologías con el fin de lograr sistemas robustos pese al tamaño o complejidad de los mismos. A la vez, la necesidad de resolver problemas del mundo real en donde se involucra la representación de objetos 3D, ha llevado al surgimiento y fortalecimiento de la computación gráfica, como rama de los sistemas computacionales. Esto se refleja en los programas de uso interactivo ante las interfaces gráficas de usuario puesto que cada vez es necesario que sean más agradables y sencillas de entender y de usar [2] [3] [4] [5].

Junto con el software ha evolucionado el hardware, en donde los diversos dispositivos de entrada y salida que apoyan los procesos gráficos permitiendo una mayor y mejor interacción, siendo así, “Las computadoras una herramienta poderosa para producir imágenes en forma rápida y económica. De hecho, no existe área alguna en la que no se pueda aplicar la computación gráfica con algún beneficio, permitiendo el surgimiento continuo de nuevas aplicaciones. Actualmente, la computación gráfica

se utiliza con frecuencia es diversas áreas como las ciencias, ingeniería, arte, entretenimiento, publicidad, educación y capacitación, entre otras" [1].

En la actualidad, la evolución en el desarrollo de sistemas ha traído como consecuencia varios problemas o dificultades para la computación gráfica, entre los cuales se encuentran: poca reusabilidad del código, distribución de archivos, dificultad en el despliegue de aplicaciones y casi imposible la automatización de estas, gran dependencia de las librerías gráficas y el constante versionamiento de las aplicaciones [7] [8] [9] [10].

Una alternativa tecnológica que ha surgido en los últimos años para minimizar estas deficiencias en el desarrollo de software en general, es el uso de un contenedor de aplicaciones, el cual está definido como un administrador del ciclo de vida de los elementos que alberga, ya sean clases, componentes o librerías, logrando un desacoplamiento entre la lógica del negocio y los requerimientos adicionales (cualidades del sistema) [7] [10].

En este artículo se presenta en el numeral 2 el diseño arquitectónico de un contenedor de aplicaciones con características específicas, que permiten albergar aplicaciones de computación gráfica. En el numeral 3 se presenta un prototipo funcional de la arquitectura y como se encaja en la arquitectura Vitral. En el numeral 4 se presentan los diferentes modelos de programación que se pueden alcanzar usando el contenedor. Finalmente, en el numeral 5 se muestra uno de los casos de validación de la arquitectura usando el modelo ATAM (Architecture Trade-Off Analysis Method) [11] [12].

2 Arquitectura del contenedor de aplicaciones

Al ser el contenedor de aplicaciones un orquestador en este caso de servicios de computación gráfica, su arquitectura se compone de módulos basados en requerimientos no funcionales o cualidades del sistema. La figura 1 muestra los 6 módulos de los que esta se compone: Container Launcher, Descriptor, Entities, Exception, Integration y Log.

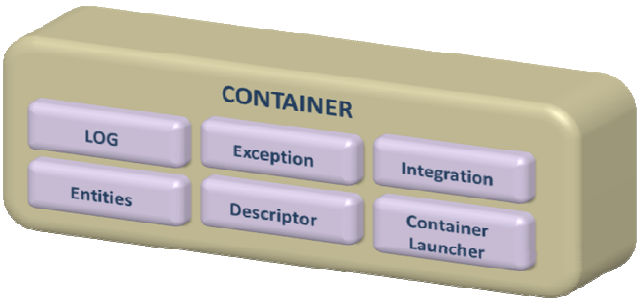

Fig. 1 Arquitectura del contenedor

- **Lanzador del Contenedor** (Container Launcher): es el encargado de gestionar los otros módulos que se encuentran en el contenedor, actuando como un

coordinador, con el fin de inicializarlo y darle ejecución, para esto se encarga de configurar el contenedor, cargar y registrar los servicios y las aplicaciones, y por último de inicializar el registro de la información que estará disponible para los usuarios en los Logs.

- **Entidades** (Entities): interactúa con el módulo descriptor ayudando a buscar los servicios de computación gráfica que se están ofreciendo, también se relaciona con el módulo de integración en el momento de buscar servicios dinámicos y contextos externos al contenedor. En este se reúnen todas aquellas entidades que son importantes para el contenedor, contiene la información relevante de los servicios, de las aplicaciones y del mismo contenedor. La información relevante que se almacena en las entidades es: (1) en los servicios el nombre del servicio, lugar del despliegue, nombre del archivo de extensión .JAR, lugar de implementación del servicio; (2) en las aplicaciones nombre de la aplicación, servicios que utiliza del contenedor, lugar de despliegue, nombre del archivo de extensión .JAR, lugar de implementación de la aplicación. Y (3) en el contenedor las configuraciones básicas, entre las que está el puerto y configuración del registro de información.
- **Integración** (Integration): es el encargado del manejo del contexto del contenedor y de los servicios dinámicos. Registra los servicios dinámicos que son identificados por medio del módulo descriptor, y luego busca contextos externos para ser integrados con los servicios que el contenedor ofrece. Lo primero que este módulo realiza es la creación del contexto, esto con el fin de que se puedan registrar los servicios dinámicos, los diferentes contextos externos y más adelante las aplicaciones y servicios estáticos.
- **Descriptor** (Descriptor): contiene los diferentes descriptores que el contenedor requiere, esto con el fin de obtener información de servicios (dinámicos y estáticos), aplicaciones y del contenedor, cuando el descriptor encuentra la información la lee por medio del módulo XML que se encuentra en la capa de servicios y luego la transmite para ser encapsulada en entidades. Los descriptores también se encargan de validar la información mínima para satisfacer los parámetros requeridos por el módulo de entidades.
- **Registros del Contenedor** (Container Logs): Un log es un registro que se lleva de información determinada y de carácter importante para un uso posterior. Este módulo configura todos los logs que tiene el contenedor, dando soporte a diversos tipos, entre estos están: el log de consola, el log del contenedor, el log de servicios dinámicos, el log de servicios y el log de errores. Estos logs solo tienen dos formas de ser mostrados al usuario, por consola y por archivos.
- **Excepciones** (Exception): es el encargado de gestionar las respuestas indicadas ante las distintas excepciones que se pueden presentar en el contenedor.

Estos módulos forman la capa del contenedor, en donde se puede interactuar tanto con la capa superior, en este caso la capa que recibe e interpreta las solicitudes y respuestas y la capa inferior, cuya responsabilidad es almacenar todos los servicios funcionales que se le brindan al usuario.

3 Prototipo Funcional del Contenedor de Aplicaciones

Para darle funcionalidad al modelo arquitectónico se desarrollo un prototipo el cual se incrustó como una capa más en la arquitectura de Vitral. (Arquitectura de computación gráfica desarrollada por el grupo de investigación "Takina", de la Pontificia Universidad Javeriana) [6].

El modelo arquitectónico de Vitral está definido a través de capas, las cuales son: Application, Presentation, Service, VSDK y Librerías Graficas, todas soportadas por el JDK (Java Development Kit). La Fig. 2. Muestra el modelo arquitectónico incluyendo el contenedor de aplicaciones como capa de la arquitectura.

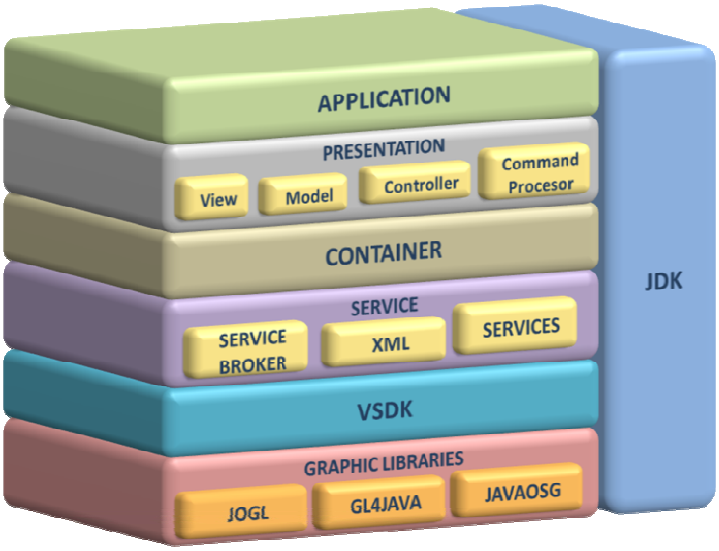

Fig. 2 Arquitectura de Vitral con Contenedor

• **Capa de aplicación**: La capa de aplicación es la de más alto nivel de abstracción, puesto que es la que interactúa directamente con el usuario por medio de la creación o uso de aplicaciones que este crea necesaria. El objetivo es brindar todos los servicios posibles al programador con el fin que pueda construir una aplicación orientada a computación gráfica con la menor cantidad de líneas de código posible.
• **Capa de Presentación**: esta implementada por el patrón Modelo-Vista-Controlador (MVC) [3], donde, la información al usuario se despliega por medio de interfaces ya sean en código o en imágenes (Vista); se administra e interpreta el manejo de los dispositivos de interacción (Controlador) y la conexión entre las solicitudes y los servicios (Modelo). Con esto la capa de presentación hace el papel de intermediario entre las aplicaciones del usuario las cuales emiten las solicitudes de los servicios y la capa de servicios que atiende las solicitudes emitidas.
• **Capa de Servicios**: Esta capa contiene todos los servicios que son ofrecidos a los usuarios, en este caso están especializados en computación gráfica. Los servicios se

encuentran divididos en 3 funcionalidades diferentes o módulos. La primera es la encargada de los servicios de computación gráfica en donde se encuentran respuestas a las solicitudes como crear escenas que se construyen con grafos de escenas los cuales contienen objetos tanto en 2D como en 3D, comportamientos, deformaciones o transformaciones, también se ofrece el servicio de render y simulación para una secuencia de escenas dando la opción de adicionar efectos multimedia como sonido o video. El otro tipo de servicio es conocido como Service Broker el cual es el encargado de registrar todos los servicios que el sistema ofrece a los usuarios. Por último esta el servicio de XML, el cual es activado cuando hay despliegue de los servicios de computación gráfica.

• **Capa de VSDK**: "Vitral Software Development Kit". puede definirse como un Toolkit que da soporte a los servicios de computación gráfica y suple algunos requerimientos por medio de la interacción que tiene con la capa de librerías gráficas, permitiendo así, ser extensible y mantenible en los desarrollos de aplicaciones.

• **Capa de Librerías Gráficas**: Esta capa reúne las librerías gráficas que normalmente usan los desarrolladores de computación gráfica. Y estas permiten la generación de imágenes por medio de modelos matemáticos y características (textura, iluminación, ángulos, profundidad, entre otras) asignadas a la imagen, siendo manejadas a un bajo y complejo nivel de programación.

Una vez definido el modelo arquitectónico de VITRAL se adaptó la capa del contenedor, quedando entre la capa de presentación y la capa de servicios, como se muestra en la figura 2, de manera que recibe las solicitudes de los usuarios y orquesta los servicios que VITRAL ofrece, dando la opción de servicios distribuidos, si el programador lo desea, sin la necesidad de entrar a configurar o preocuparse por las conexiones requeridas.

Al incluir el contendor de aplicaciones en la arquitectura de vitral se generaron cambios en la interacción de las capas que constituyen el enfoque arquitectónico. La capa de presentación interactúa con el contenedor enviándole las solicitudes y recibiendo las respuestas a estas. En la capa de Servicios el Service Broker tiene la particularidad que interactúa con el contenedor en el momento del desarrollo de las aplicaciones, ya que actúa como fachada de los servicios, puesto que estos están desplegados en el contenedor. Por último está el servicio de XML, el cual es activado cuando hay despliegue de los servicios interactuando de esta manera con el contenedor, al leer la información de los archivos XML y transmitiéndola al descriptor del contenedor.

4 Modelos de programación

Por medio de los modelos de programación se explica como el software se ejecuta en una red de computadores, o nodos de procesamiento, con esto diversos elementos como redes, procesos, tareas y objetos tienen que ser mapeados a los distintos nodos que conforman la arquitectura. El mapeo del software a los nodos, necesita ser flexible y tener un mínimo impacto sobre el código fuente [11]. Por ello, se esquematiza en 2 modelos, con y sin contenedor.

4.1 Modelo de Programación sin Contenedor de Aplicaciones

El modelo de programación sin contenedor (Fig. 3) puede reconocer como la arquitectura de sistemas Stand-Alone la cual se caracteriza por ser de un solo nivel, no se integra con el contenedor y los servicios se prestan como librerías, siendo administrados por el programador en el momento de compilar la aplicación. Para este modelo solo es necesario contar con un nodo (computador) y las librerías que se van a necesitar.

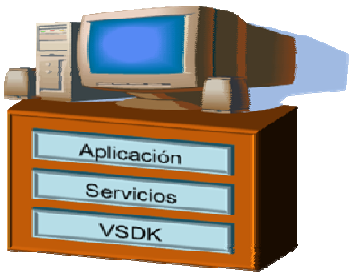

Fig. 3 Modelo de programación sin contendor

4.2 Modelo de Programación con Contenedor de Aplicaciones

Este modelo de programación implementa la arquitectura de sistemas Cliente-Servidor, la cual puede tener diferentes niveles (Fig. 4). Como factor común existe (1) la aplicación contenedor la cual puede verse como la unión o consolidación de las capas de la arquitectura de Vitral con contenedor en un solo paquete y (2) la aplicación cliente, la cual puede verse como el mecanismo que interpreta las solicitudes del programador y despliega las respuestas a estas por medio de la interacción que tiene con la aplicación contenedor. Con ello, los dos modelos son:

- **Cliente-Servidor Centralizado**: este modelo se caracteriza por contener una sola maquina la cual alberga tanto la aplicación cliente como la aplicación contenedor.
- **Cliente-Servidor Distribuido**: cuando el programador requiere un servicio, como el render de forma distribuida, este modelo es el indicado. El nodo servidor ya no realiza el procesamiento de las solicitudes, sino la administración de estas por medio de un contenedor administrador, el cual es el encargado de establecer la distribución de las tareas a los otros nodos; estas máquinas también contienen contenedores que son conocidos como contenedores esclavos. El modelo se compone de 2 o 3 niveles comunicados por el estándar RMI-IIOP, en donde, el modelo de 2 niveles se compone de un nodo cliente (aplicación cliente), otro nodo servidor (aplicación contenedor, encargada de ejecutar las solicitudes) y en el modelo de 3 niveles existe un nodo cliente (aplicación cliente), un nodo servidor (aplicación contenedor administrador) y uno o más nodos (contenedores esclavos).

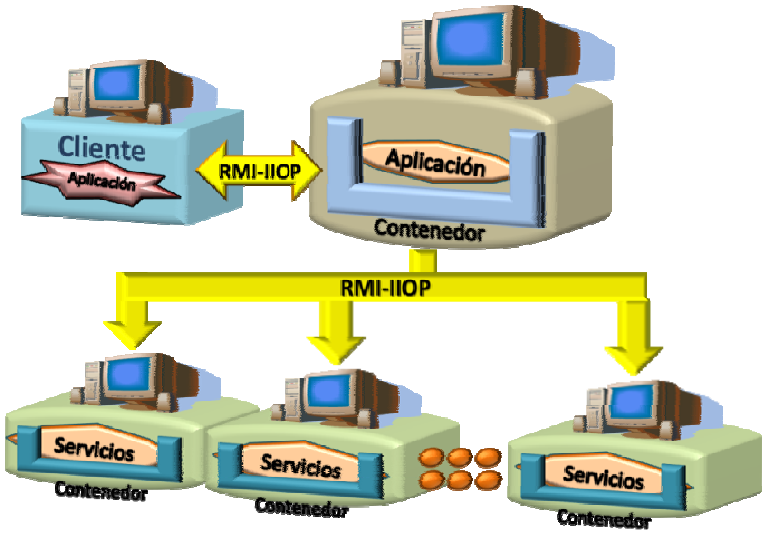

Fig. 4 Modelo de programación con contendor de 3 niveles

5 Validación de la Arquitectura

La validación de la arquitectura se realizó por medio del Método de Análisis y Compensación de la Arquitectura (ATAM, por su siglas en ingles) [12], el cual es un proceso para la mitigación de riesgos en el ciclo de vida del desarrollo de software, ayudando a determinar los puntos sensibles de la arquitectura por medio de los atributos de calidad o requerimientos no funcionales. Este método, fue desarrollado para proporcionar principios que permitan evaluar una arquitectura de software con respecto a diferentes atributos de calidad que compiten, como lo son la modificabilidad, seguridad, rendimiento, disponibilidad, etc. Estos atributos interactúan con fin de ayudar con la decisión arquitectónica. ATAM es un modelo de diseño en espiral: primero se postulan arquitecturas candidatas seguidas por un análisis y mitigación del riesgo, el cual conduce al refinamiento de la arquitectura del sistema [12] [13].

Como atributos de calidad para la validación de la arquitectura se adoptó el estándar ISO/IEC 9126-1 [13] [14], el cual define los atributos de calidad como características que a la vez se consolidan con otros atributos de calidad llamados sub-características.

Dentro de la validación se determinaron 11 escenarios o casos de prueba, los cuales están directamente relacionados con las características y sub-características del estándar. Cada escenario tiene asociado un Id, el nombre del escenario, el requerimiento funcional del contenedor, el atributo de calidad, la descripción del ambiente de prueba, un estimulo, la métrica y por último el resultado el cual permite determinar el cumplimiento del escenario. En la Tabla 1 se muestran los escenarios y el resultado obtenido en la validación de la arquitectura. Existen varios escenarios con

el mismo nombre pero las condiciones del ambiente de prueba cambian haciendo que los resultados no sean iguales.

Tabla 1 Escenarios Validados en la Arquitectura

ID Escenario	Escenario	Resultado
E001	Agregar servicios al contenedor	Se satisface
E002	Agregar servicios al contenedor	Se satisface
E003	Agregar servicios al contenedor	No se satisface
E004	Detener una aplicación en tiempo de ejecución	Se satisface
E005	Hacer invocaciones remotas de las aplicaciones	Se satisface
E006	Hacer invocaciones remotas de las aplicaciones	Se satisface
E007	Agregar aplicaciones en tiempo de ejecución	No se satisface
E008	Agregar aplicaciones en tiempo de ejecución	Se satisface
E009	Agregar aplicaciones en tiempo de ejecución	No se satisface
E010	Tener múltiples servicios en el contenedor	Se satisface
E011	Tener múltiples servicios en el contenedor	Se satisface

Por ejemplo, la respuesta al escenario E001 fue satisfactoria ya que el escenario se cumplió ante el atributo de confiabilidad. La Fig. 5 refleja la interacción entre los componentes que hacen posible el registro de un servicio en el contenedor teniendo como ambiente la ejecución normal de este, y siendo el primer servicio añadido en el contenedor.

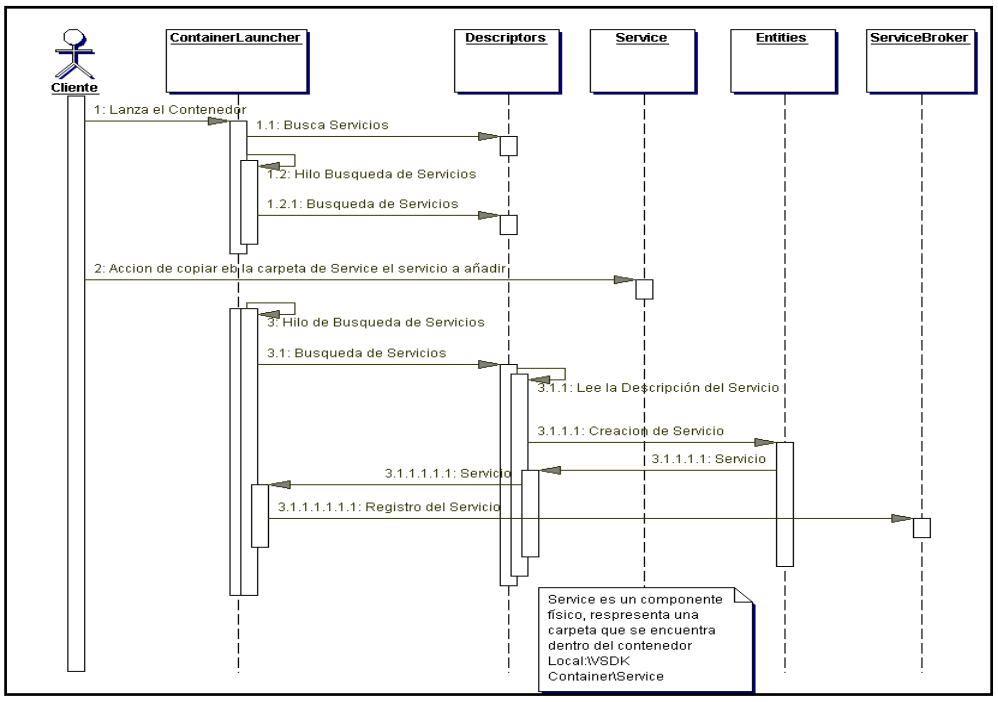

Fig. 5 Trazabilidad del escenario E001

Otro ejemplo de la validación, es el escenario E004, donde la respuesta a este escenario fue satisfactoria puesto que se cumplió con el atributo de calidad indicado. La figura 6 es la interface gráfica de los servicios prestados por el contenedor, para este caso la pestaña de aplicaciones muestra las opciones que se brindan, refrescar, permite ver las aplicaciones que recientemente se han adicionado y remover para retirar la aplicación deseada del contenedor, cumpliendo de esta manera con el escenario y el atributo de calidad de confiabilidad.

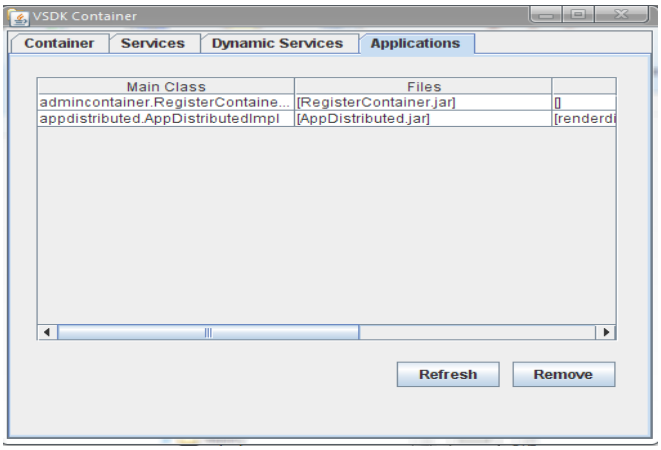

Fig. 6 Interface del contenedor

6 Conclusiones

La inclusión de un contenedor de aplicaciones a la arquitectura Vitral, es una nueva propuesta para el desarrollo de aplicaciones ofreciendo un conjunto de servicios comunes, ya sea de requerimientos funcionales como no-funcionales. El contenedor no solo es una nueva manera de desarrollar aplicaciones, sino también un nuevo campo para seguir investigando y así ampliar la arquitectura y los diferentes servicios, de manera que se dé robustez tanto a la arquitectura como al contenedor. La inclusión de un contenedor de aplicaciones permite que el desarrollo de nuevas aplicaciones este enfocada a la parte funcional, sin necesidad de preocuparse por los servicios que se requieran.

El contenedor ofrece a los programadores un ambiente más flexible e integrado por medio de una capa de alto nivel de abstracción en el desarrollo de aplicaciones. Por ello, se considera que esta primera aproximación, es un gran paso que se debe continuar con el fin de mejorarla y ofrecer más servicios y soluciones a los desarrolladores.

Más que establecer que el contenedor ofrece facilidad en el desarrollo de aplicaciones, se destaca la orquestación que éste hace a los servicios de computación gráfica, respondiendo por las tareas que antes el desarrollador debía llevar a cabo.

Dentro de estas tareas se encuentran: configuraciones básicas de la escena, cámara, tipo de render a utilizar y la interacción dentro de la escena.

La extensibilidad y mantenibilidad de la arquitectura propuesta, se refleja en la posibilidad de incluir e interactuar con nuevas librerías gráficas, servicios, módulos con nuevas funcionalidades para el contenedor y nuevas aplicaciones, manteniendo la transparencia ante los usuarios. Además se ofrecen recursos computacionales distribuidos, brindándole a los desarrolladores la facilidad de distribuir escenas e integrarlas con servicios de multimedia generando así no solo imágenes sino videos.

Como servicio distribuido se implementó el proceso de render, brindándole a los desarrolladores la facilidad de distribuir la escena con algoritmos distribuidos, para este caso se da soporte al algoritmo de Sort Last. Con esto se deja como trabajo futuro la implementación del servicio de simulación y la funcionalidad de simulación distribuida.

Referencias

1. V. N. C. Camacho. (2005). Introducción a la Computación Gráfica. [Online]. Disponible: http://www.inf.unitru.edu.pe/~vncc/Cursos.html
2. Angel, E., (2000). Interactive computer graphics: a top-down approach with OpenGL, Addison Wesley.
3. Lange, J.,(1984). Design Dimensioning with Computer Graphics Applications, CRC Press.
4. Cunningham, S.,(2006). Computer Graphics: Programming in OpenGL for Visual Communication, Pearson Prentice Hall.
5. Govil-Pai, S.,(2004). Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya, Springer.
6. Pontificia Universidad Javeriana. (2007, Ene, 21) Vitral SDK Toolkit documentation. [Online]. Disponible: http://vitral.sourceforge.net/html_doxygen/
7. Gobel, S.; Pohl, C., (2004). Aigner, R.; Pohlack, M.; Rottger, S. & Zschaler, S., The COMQUAD Component Container Architecture, IEEE.
8. Hallstrom, J.; Sridhar, N.; Sivilotti, P.; Arora, A. and Leal, W.,(2002). A Container-Based Approach to Object-Oriented Product Lines, Journal of object technology.
9. Hatcherson, R.; Holt, K. and Tarter, S.,(2004). A Container-Based Architecture for Simulation.
10. Sridhar, N. and Hallstrom, J.,(2006). A Behavioral Model for Software Containers, Springer Berlin / Heidelberg.
11. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture (POSA 1). Wiley and Sons, 1996
12. R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The Architecture Tradeoff Analysis Method. Proceedings of ICECCS '98, (Monterey, CA), Agosto 1998, pp. 68-78.
13. F Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif. Quality Characteristics for Software Architecture Journal Of Technology. 2003
14. M. Barbacci, M. Klein, T.A. Longstaff, and C. Weinstock. Quality Attributes. Carnegie Mellon University, 1995