UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

TÍTULO: ARQUITECTURA PARA EL SISTEMA DE REALIDAD AUMENTADA KINETIC POSTURAL.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

AUTORES:

Dayrel García Reynaldo.

Fabián Villegas Hernández.

TUTORES:

Ing. Ernesto de la Cruz Guevara Ramírez.

Ing. Mileydi Moreno Mirabal.

Cuidad de La Habana, julio del 2010

"La imaginación es más importante que el conocimiento. El conocimiento es limitado, mientras que la imaginación no"

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los _____ días del mes de _____ del año _____

_____ _____

Dayrel García Reynaldo Fabián Villegas Hernández.

Autor Autor

_____

Ing. Ernesto de la Cruz Guevara Ramírez

Tutor

_____

Ing. Mileydi Moreno Mirabal

Tutor

AGRADECIMIENTOS

De Fabián:

A todas las personas que han colaborado de una forma u otra en la realización de este trabajo, y que de manera directa o aún sin saberlo han ayudado en mi formación tanto personal como profesional. A todas mis amistades que vienen conmigo desde el pre y a los amigos de la Universidad, a nuestros tutores Ernesto y Mileydi, por su ayuda y apoyo.

De Dayrel:

Quisiera agradecer a mi mamá por apoyarme y siempre estar presente en todos los momentos difíciles dándome su cariño, por fomentar en mí el espíritu de superarme y luchar por mis sueños, por la confianza que siempre ha depositado en mí.

A mi tía Raquel por siempre estar presente y quererme tanto, por ser mi otra mamá, a mis tíos Alberto y Tomás por su apoyo y sus consejos para siempre salir adelante.

A mis primas Geldita, Pilar y Neisy, a mis primos Mario y Vladimir a todos ellos por su ayuda incondicional.

A mi papá Nivaldo, a mis hermanos Luis, Eduardo y Noly, así como a toda esta parte de la familia, que son muchos pero todos me han ayudado desde que nos conocimos.

A mis amigos del barrio y del pre a pesar de estar lejos, siempre han estado y presente y me han dado fuerza para continuar.

Aquí en la escuela agradecer a mis amigos que me han ayudado, dígase Addsel, Jabier, Yosuan, Ihordan, Pablo, Yenia y Yamilet, a la gente del aula vieja y a la actual, al personal de mi apartamento y a todos los que me ayudaron.

A Fabián por ser mi compañero de tesis y su familia por su apoyo incondicional.

A todas las personas que han colaborado de una forma u otra en la realización de este trabajo, y que de manera directa o aún sin saberlo han ayudado en mi formación tanto personal como profesional.

A Ernesto y Mileydi, por su ayuda y apoyo incondicional en todo momento.

A todos muchas gracias

DEDICATORIA

De Dayrel:

Quisiera dedicar este trabajo a mi familia.

A mi mamá por su ejemplo y dedicación.

A mi tía por ser mi segunda mamá.

A mis tíos por su apoyo incondicional.

A mis primos (a)s, a mis hermanos y a mi padre por su ayuda y sus consejos.

A mis amigos del barrio y a mis compañeros de aula, aquel grupo que 5 años atrás fuera el 5105 y a mi grupo actual 5504 por acogerme.

A mi compañero de tesis…

De Fabián:

En primer lugar a mi mamá y mi papá, él por ser mi ejemplo en todo momento y a ella por sus consejos, a mi hermana Beatriz que la quiero mucho, a mis sobrinos Armando J. y Mónica que también los quiero, Armando que es el mejor cuñado que hay, a mi tía Ana Gloria por preocuparse por mí, a mi primo Javier para que le sirva de ejemplo y siga estudiando y esforzándose a ser mejor, a mi amigos Jadier y Rebeca, Jadier que dentro de poco nos separaremos y quiero que me recuerde y Rebeca que es muy especial para mí, a Tony por su apoyo, a mis abuelos Gloria y José "Che" por ser los mejores conmigo, a mi familia de Camagüey, mis abuelos, tíos, hermana, sobrina y primos. A mis compañeros que vienen conmigo desde primer año, entre ellos a mi compañero de tesis por aguantarme todo este tiempo y a toda las personas que no pueden estar aquí por un motivo o por otro.

RESUMEN

En el presente trabajo de diploma se describe una arquitectura de software basada en componentes que puede ser utilizada como referencia en el desarrollo de aplicaciones de realidad aumentada, por tal motivo uno de los objetivos fundamentales es la descripción y evaluación de la arquitectura. Para alcanzar los objetivos planteados se ha centrado la investigación en el estudio de los conceptos relacionados con la arquitectura de software, con el fin de lograr una arquitectura que garantice buenas prácticas de programación, soluciones robustas que carezcan de complejidad innecesaria y con mejoras en propiedades como la escalabilidad y la reusabilidad.

Palabras Clave:

Arquitectura de Software, Patrón, Estilo Arquitectónico, Métodos de Evaluación.

SUMMARY

With this investigation we described a software architecture based on components that can be used as a reference in the development of applications of Augmented Reality, for that reason one of the main objectives is the description and evaluation of the architecture. To reach the proposed objectives, the investigation has been focus at the study of the concepts related to the software architecture, with the final goal of develop an architecture that guarantee good programming practices, strong solutions with no unnecessary complexities and with improvements at properties like reusability and scalability.

Key Words:

Software architecture, patterns, Architectural style, Evaluation methods.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En los últimos años con el desarrollo de las tecnologías de la información los Sistemas de Realidad Virtual (SRV) han experimentado una serie de transformaciones, las que el hombre ha tenido que subdividir en pequeñas ramas para su mejor comprensión, estudio y utilización. Una de las categorías más importantes es la Realidad Aumentada (RA), esta tecnología combina los datos virtuales con el ambiente real observado por el usuario.

La RA se encuentra en un proceso emergente y en franco ascenso; es una visualización directa o indirecta en tiempo real de un entorno físico en el mundo real, cuyos elementos se combinan (o aumentan) mediante dispositivos digitales (lentes, webcam, móviles, etc.) añadiendo información virtual a la información física ya existente. La principal diferencia con la Realidad Virtual (RV), puesto que no sustituye la realidad física, sino que sobre pone los datos informáticos al mundo real.

Los sistemas de RA son usados para añadir gráficos generados por una computadora, en tiempo real, a la escena real que alguien observa. Los objetos virtuales son colocados en la escena de tal manera que la información que se expone en dichos objetos aparece en el lugar correcto con respecto a los objetos reales.

La RA intenta llevar la relación entre los humanos y las computadoras a un nivel donde la información virtual no se encuentre en una pantalla, sino tratar de llevar esa relación a un nivel más real.

La RA ofrece nuevas posibilidades de interacción que pueden ser usadas en una gran variedad de áreas como se manifiesta a continuación:

- Proyectos educativos: Exhibiciones virtuales en museos, o atracciones en parques de diversión y mejoramientos en las metodologías educativas y procesos de evaluación
- Entretenimiento: Se pueden aportar grandes innovaciones en el campo de los videojuegos, una industria que crece cada vez más con el paso del tiempo.
- Simulación: Tanto de automóviles, como de aviones, y de otros medios de transportes.
- Servicios de emergencias y militares: En una emergencia se podrían mostrar las rutas óptimas para llegar a las salidas de emergencia, así como, las instrucciones que se debe de hacer en

dichas situaciones. En el campo militar, podría servir para mostrar información de mapas, de localización de enemigos y/o puntos de reunión, etc.

- En la medicina: Gracias a esta tecnología se pueden observar los tejidos del cuerpo mediante una ampliación virtual del organismo, sin necesidad de recurrir a los rayos X, mejoramiento de procesos y entrenamiento de cirugías.

- Procesos Industriales: Entrenamiento de operarios, asistencia de operarios. Creación, Producción y Control de calidad de productos, arquitectura, automotriz y metalurgia.

A escala mundial los Sistemas de Realidad Aumentada (SRA) se utilizan diferentes tipos de arquitecturas las más utilizadas son:

- Arquitectura 3 Capas: su objetivo primordial es la separación de la capa de presentación, capa de negocio y la capa de datos. Sus principales ventajas son, la reutilización de capas, facilita la estandarización, las dependencias se limitan a intra-capa y contención de cambios a una o pocas capas.

- Arquitectura Orientada a Servicios (en inglés Service Oriented Architecture): es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio. Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización. Entre sus ventajas tenemos la mejora en los tiempos de realización de cambios en procesos, facilidad para evolucionar a modelos de negocios basados en tercerización y la facilidad para abordar modelos de negocios basados en colaboración.

- Modelo Vista Controlador (MVC): es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

En la Universidad de las Ciencias Informáticas, en el Centro de Desarrollo de Informática Industrial, el proyecto de Laboratorio de Realidad Aumentada (ARLab) realiza en conjunto que el Centro Nacional de Rehabilitación “Julio Díaz” un proyecto en el cual se espera desarrollar un producto que lleva por nombre

"Kinetic Postural". En el proyecto ARLab no están identificados los elementos estructurales que van a componer el sistema y sus interfaces, no se ha definido la composición de estos elementos así como su comportamiento en subsistemas progresivamente más grandes, también en el proyecto se necesita describir que estilo arquitectónico va a guiar el proceso. De ahí que surja el problema científico: ¿Cómo organizar un sistema de software funcional, flexible, reutilizable y de fácil comprensión para todos los involucrados en el proyecto? Para darle solución a este problema se plantea el siguiente objetivo: diseñar una arquitectura que sea capaz de describir los elementos y colaboraciones que van a conformar el producto.

La investigación tiene como objeto de estudio la arquitectura de software, constituyendo el campo de acción la arquitectura para SRA basados en monitores. Para darle solución al objetivo de la investigación se trazaron las siguientes tareas investigativas:

- Analizar las tendencias actuales de los estilos arquitectónicos para identificar los más utilizados en los SRA.
- Identificar patrones arquitectónicos y sus ventajas para describir la arquitectura del software.
- Seleccionar herramienta y metodología para definir una arquitectura.
- Seleccionar uno a varios lenguajes de modelado para describir la arquitectura.
- Diseñar una arquitectura fiable que cumpla con los requerimientos de un sistema de Realidad Aumentada.

En el trabajo utilizan los siguientes métodos investigativos nombrados a continuación:

- Analítico-Sintético: Se analizan una serie de documentos y teorías que permitieron extraer los elementos más importantes que se relacionan con el objeto de estudio.
- Inductivo-Deductivo: Este método se utiliza para darle solución al problema planteado a través de conocimientos generales sobre el tema de investigación, llegando así a conclusiones particulares.
- Análisis histórico lógico: Se emplea para consultar el desarrollo y evolución de la arquitectura de los diferentes SRA en Cuba y el mundo.

La tesis está estructurada en tres capítulos de la siguiente manera:

- **Capítulo 1: Fundamentación teórica.**

En este capítulo se abordarán los elementos relacionados con el estudio de las arquitecturas, principales herramientas y metodologías de desarrollo más comunes para determinar cuál es la adecuada para desarrollar el proyecto antes de iniciar el trabajo, apoyando la investigación en el estudio de los SRA basado en monitores.

- **Capítulo 2: Descripción de la arquitectura.**

En este capítulo se propone una arquitectura que se describe a través de las vistas arquitectónicas, considerando para ello los requerimientos que debe cumplir el sistema.

- **Capítulo 3: Evaluación de la arquitectura.**

En este capítulo se analizan los métodos de evaluación y atributos de calidad más adecuados de acuerdo a las características del sistema y se selecciona el más apropiado para evaluar la arquitectura. Se muestran los resultados obtenidos de la evaluación después de aplicar el método propuesto.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se hace un estudio de los temas relacionados con la Arquitectura de Software (AS), estilos arquitectónicos, patrones de diseño y arquitectónicos, lenguajes, herramientas de modelado y su relación con los SRA.

1.1 Arquitectura de software.

La arquitectura de software se centra tanto en los elementos estructurales significativos del sistema, como subsistemas, clases, componentes, nodos y las colaboraciones que tienen lugar entre estos elementos a través de las interfaces. Los casos de uso dirigen la arquitectura para hacer que el sistema proporcione la funcionalidad y uso deseado, alcanzando a la vez objetivos de rendimientos razonables. Una arquitectura debe ser completa, pero también debe ser suficientemente flexible como para incorporar nuevas funciones, y debe soportar la reutilización del software existente. (Ivar Jacobson, 2000)

Una definición reconocida es la de Clements: “La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.” (Clements, 1996)

La definición más usada de AS es de la IEEE Std 1471-2000, que plantea: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

En el libro "An introduction to Software Architecture", David Garlan y Mary Shaw definen que la Arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema". (David Garlan, 1994)

La arquitectura abarca decisiones importantes sobre:

- La organización del sistema.

- Los elementos que componen el sistema y sus interfaces, unido a su comportamiento.
- La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- Estilos de arquitectura que guían esta organización: elementos y sus interfaces, sus colaboraciones y su composición.

Un sistema de software grande y complejo requiere una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común y para esto se necesita:

1. Comprender el sistema: Para que un proyecto desarrolle un sistema, dicho sistema debe ser comprendido por todos los que vayan a intervenir en él. Lograr que los sistemas modernos sean comprensibles es un reto importante por muchas razones:
    - Abarcan un comportamiento complejo.
    - Operan en entornos complejos.
    - Son tecnológicamente complejos.
    - A menudo, combinan computación distribuida, productos y plataformas comerciales y reutilizan componentes y marcos de trabajo.
    - Deben satisfacer demandas individuales de la organización.
2. Organizar el desarrollo: Cuanto mayor sea la organización del proyecto de software, mayor será la sobrecarga de comunicación entre los desarrolladores para intentar coordinar sus esfuerzos. Dividiendo el sistema en subsistemas, con las interfaces claramente definidas y con un responsable o un grupo de responsables establecido para cada subsistema, el arquitecto puede reducir la carga de comunicación entre los grupos de trabajo de los diferentes subsistemas.
3. Fomentar la reutilización: Permite la reutilización de componentes y otros elementos en otros proyectos similares permitiendo así menor costo en tiempo.
4. Hacer evolucionar el sistema: El sistema debe ser en sí mismo flexible a los cambios o tolerante a los cambios, debe ser capaz de evolucionar sin problemas puesto que las arquitecturas de los

sistemas pobres suelen degradarse con el paso del tiempo y necesitan ser parcheadas hasta que al final no es posible actualizarla con un coste razonable. (Ivar Jacobson, 2000)

1.1.1 Arquitectura de software basada en componentes.

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado. El estilo de arquitectura basado en componentes tiene las siguientes características:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

1.1.2 Desarrollo de Software basado en Componentes.

El desarrollo de software basado en componentes se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software, trata de fundamentar las bases para el diseño y desarrollo de aplicaciones distribuidas basadas en componentes de software reutilizable. Puede verse como una extensión natural de la programación orientada a objetos dentro del ámbito de los sistemas abiertos y distribuidos, este paradigma se basa en el uso de los componentes de software como entidades básicas del modelo, entendiendo por componentes “una unidad de composición de aplicaciones o sistemas de software que posee un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes, de forma independiente en tiempo y espacio” según Szyperski. Entre sus ventajas e inconvenientes se tienen:

- ✓ Ventajas:
  1. Reutilización del software. Permite alcanzar un mayor nivel de reutilización de software.
  2. Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.

3. Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
4. Mayor calidad. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

El desarrollo de software basado en componentes se basa en la reutilización de los componentes, estos se diseñan y desarrollan con el objetivo de utilizarse en otras aplicaciones, al utilizar componentes ya probados previamente reduce el tiempo de desarrollo, se mejora la fiabilidad del producto y aumenta la competitividad en coste.

1.1.3 Modelos de Procesos de desarrollo de Software.

Un sistema de software puede ser considerado desde un punto de vista de ciclo de vida. Esto significa que el sistema es observado desde la primera noción de su existencia hasta su operación y administración. Algunas diferentes aproximaciones al desarrollo del ciclo de vida son consideradas debajo. Las diferentes aproximaciones están todas basadas en las mismas actividades, y varían solo en la forma en que son interpretadas. En esta sección se da una visión general de los modelos de desarrollo del ciclo de vida del Desarrollo de Software basado en Componentes.

- ✓ El modelo de Secuencias

El modelo de secuencias (por ejemplo, el Modelo de Cascada) sigue una sistemática y secuencial aproximación que comienza en el nivel de sistema y progresa exitosamente desde análisis hasta las pruebas. Cada actividad es considerada como concluida antes de que la próxima actividad comience. La salida de una actividad es la entrada para la otra. Este acercamiento al desarrollo de software es el más viejo y el más ampliamente usado. Se basa en la noción de que es posible definir y describir todos los requerimientos del sistema y características del software de antemano, o al menos bastante temprano en el proceso de desarrollo. Muchos problemas mayores en la ingeniería de software surgen del hecho de que es difícil para los clientes definir todos los requerimientos explícitamente. Un modelo de secuencia requiere una completa especificación de requisitos y ahí pueden haber dificultades en acomodar la incertidumbre natural que existe en el comienzo de muchos proyectos. Además, es difícil agregar o modificar requerimientos durante el desarrollo porque, una vez realizado, las actividades se consideran como terminadas. En la práctica los requerimientos cambiarán y nuevas funcionalidades serán llamadas y

una aproximación puramente secuencial al desarrollo de software puede ser dificultoso y en muchos casos insatisfactorio. Otro problema que se encuentra a veces cuando se aplica un modelo de secuencia es la tardía respuesta de los clientes. Una versión que trabaje del software no estará disponible hasta tarde en el proceso de desarrollo del sistema y un mayor desconocimiento, si no se detecta hasta que el programa final sea revisado, puede ser desastroso. El modelo de secuencia tiene un importante lugar en la ingeniería de software. Aunque en la práctica nunca es usada en su forma pura, el modelo de secuencias se ha mantenido como el modelo de procesos de desarrollo de software más influyente. .

- ✓ Modelo Incremental

El modelo incremental combina elementos del modelo secuencial (aplicado repetitivamente) con la aproximación iterativa. El modelo incremental aplica el modelo de secuencias en etapas como el progreso del tiempo del calendario. Cada secuencia produce un incremento entregable del software con nuevas funcionalidades adicionadas al sistema. Cuando el modelo incremental es usado, el primer incremento es a menudo el núcleo del sistema de software. Los requisitos básicos son direccionados, pero muchas características suplementarias se mantienen sin entregar.

Este proceso es repetido siguiendo la entrega de cada incremento, hasta que el sistema de software completo sea desarrollado. El modelo incremental es particularmente útil para manejar cambios en los requerimientos. Tempranos incrementos pueden ser implementados para satisfacer requerimientos, dichos requerimientos pueden ser mejorados en incrementos posteriores. Si el núcleo del software es bien recibido, el próximo incremento puede ser adicionado de acuerdo con el plan. Los incrementos pueden ser planeados además para administrar riesgos técnicos.

- ✓ Ventajas:

1. Las iteraciones son controladas.
2. La motivación será grande ya que se cumplirán los objetivos a corto plazo.
3. Flexibilidad: No siempre habrá los mismos requerimientos hasta el final del proyecto, en cada versión habrá oportunidad para tomarlos en cuenta.
4. Retro-alimentación con los usuarios: Ayuda a mejorar el software en cada versión.

- ✓ Desventajas:

1. Hay que entregar funcionalidad en cada pequeña iteración.

2. Se necesita una extensa planeación si se piensa trabajar en versiones.

✓ Modelo de Prototipo

El modelo evolutivo para desarrollo de software está basado en la creación de un prototipo bastante temprano en las actividades de desarrollo. El prototipo es una versión preliminar, o intencionalmente incompleta, del sistema completo, el prototipo es entonces gradualmente mejorado y desarrollado hasta que alcanza un nivel de aceptación decidido por el usuario. Se trata de un ciclo de vida de software basado en componentes. Este modelo cuando:

- No se entienda la aplicación del software a desarrollar, por eso este modelo ayuda a experimentar y/o probar.
- A veces suelen terminar el contrato cuando el proceso de producción está bien avanzado, para evitar esto se realiza un prototipo de bajo costo y en corto tiempo.
- A veces no se conoce la reacción de los usuarios para con el nuevo software, entonces un prototipo ayudaría para probar diferentes diseños.

✓ Modelo Iterativo

El acercamiento iterativo al desarrollo de software está basado en el modelo secuencial suplido con la posibilidad de retornar a actividades previas. Cada iteración dirige el sistema completo e incrementa la funcionalidad de las partes del sistema. Este acercamiento permite futuros refinamientos de los requerimientos del sistema, lo que incrementa la posibilidad de administrar los requerimientos de bajo nivel. Una desventaja específica de usar este acercamiento es la imposibilidad de predecir la fecha de realización y el costo de la implementación final.

1.2 Estilos Arquitectónicos.

Se define un *estilo arquitectónico* como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes. (Frank Buschmann, 1996)

También se define como *estilo arquitectónico* a la familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su

aplicación y la composición asociada y las reglas para su construcción. Así mismo, se considera como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo. Éste incluye información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación. (Erich Gamma)

A simple vista, ambas definiciones parecen expresar la misma idea. La diferencia entre los planteamientos anteriores viene dada por la amplitud de la noción de componente en cada una de las definiciones. Buschmann asume como componentes a subsistemas conformados por otros componentes más sencillos, mientras que Shaw y Garlan utilizan la noción de componente como elementos simples, ya sean de datos o de procesos. En virtud de esto, la diferencia entre ambas definiciones gira en torno al nivel de abstracción, dado que Buschmann plantean un grado mayor en su concepto de estilo arquitectónico, sugiriendo una estructura genérica para la organización de componentes de ciertas familias de sistemas, independientemente del contexto en que estas se desarrollen.

1.2.1 Estilo arquitectónico Modelo-Vista-Controlador (MVC).

El estilo arquitectónico Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres paquetes diferentes. El modelo administra el comportamiento y los datos del dominio de aplicación. Las vistas manejan la visualización de la información. La clase controladora interpreta las acciones que se realicen en la escena, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Modelo: Es una representación detallada de la información con la que el sistema debe operar. Es en el cual la lógica de datos garantiza y asegura la integridad de los mismos permitiendo derivar nuevos datos.

Vista: Es la encargada de presentar el modelo con que se interactuará con un formato que sea el adecuado para los usuarios. Generalmente es la interfaz de usuario de la aplicación.

Controlador: Es el encargado de mediante la invocación de eventos o acciones del usuario, realizar cambios en el modelo y en algunas ocasiones hasta en las vistas.

1.3 Patrones Arquitectónicos.

Buschmann define un *patrón* como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. De manera general, un patrón sigue el siguiente esquema:

- Contexto: Es una situación de diseño en la que aparece un problema de diseño.
- Problema: Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- Solución: Es una configuración que equilibra estas fuerzas. Esta abarca:
    - Estructura con componentes y relaciones.
    - Comportamiento en tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc (Frank Buschmann, 1996)

Partiendo de esta definición, propone los *patrones arquitectónicos* como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma en que estos colaboran entre sí.

Los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos.

Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación con amplitud de todo el sistema y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un software.

1.4 Estilos y Patrones arquitectónicos.

En algunas bibliografías referentes al estudio de los patrones arquitectónicos se suele llamar de la misma manera a los patrones y estilos, pero la mayoría de los autores los ven de manera separada. Ambos se refieren a formas de estructurar los sistemas y cómo relacionar los componentes de estos, la diferencia radica en el nivel de abstracción en que se aplican. Los estilos están en un plano de abstracción más alto, definiendo cómo configurar la arquitectura, mientras los patrones están más cercanos al diseño, incluso podría decirse que más cercano a algo físico, pues estos pueden representarse mediante código en un

lenguaje de programación determinado. Los elementos por los que difieren son mostrados en la siguiente tabla.

Elemento a comparar	**Estilo Arquitectónico**	**Patrón Arquitectónico**
Definición	Son una categorización de sistemas, además describen sistemas completos.	Son soluciones generales a problemas comunes, además describen parte del sistema.
Representación	Sólo describe el esqueleto estructural general para aplicaciones.	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de una aplicación.
Dependencia de contexto	Son independientes del contexto al que puedan ser aplicados.	Partiendo de la definición de patrón, requieren de la especificación de un contexto del problema.
Solución	Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño.	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta.

Tabla 1: Diferencias entre Estilos y Patrones.

Una vez que se analizaron los elementos esenciales de los estilos y patrones de arquitectura, se concluye que existen convergencias entre ambos conceptos, hay que tener en cuenta que los patrones se refieren más a prácticas de reutilización mientras que los estilos conciernen a teorías sobre la estructura de los sistemas, por tanto, son términos diferentes en dependencia del nivel de abstracción que se tenga en cuenta.

1.5 Patrones de Diseño

Un *patrón de diseño* provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular. Son menores en escala que los patrones arquitectónicos, y tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema, debido a que especifica a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema. (Sistema de posicionamiento en la creación de un libro interactivo, Revista Digital Universitaria)

Un patrón de diseño de manera general no es más que una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios; un proyecto o estructura de implementación que logra una finalidad determinada. Es una manera más práctica de describir ciertos aspectos de la organización de un programa, conexiones entre componentes de programas y la forma de un diagrama de objeto o de un modelo de objeto.

Los patrones de diseño son soluciones a problemas comunes de diseño en un determinado contexto, y que están presentes en el desarrollo de software. Permiten formalizar un vocabulario común entre los diseñadores del sistema y estandarizar el modo en se realiza el diseño. Su utilización permite entender de manera fácil, de mantener y ampliar el sistema. Contribuye a la reutilización y al diseño de componentes de software, a la organización del código, a la flexibilidad y extensibilidad, y la facilidad de realizar cambios en el sistema. Tratan los problemas de diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Son soluciones exitosas a problemas comunes basadas en la experiencia y que se ha demostrado que funcionan.

El uso de los patrones de diseño no es de carácter obligatorio, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. De hecho, forzar el uso de los patrones puede ser un error.

Los patrones GRASP (General Responsibility Assignment Software Patterns) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software. Dentro de los patrones GRASP se pueden distinguir cinco patrones principales y cuatro adicionales.

GRASP principales:

Experto.

Creador.

Alta Cohesión.

Bajo Acoplamiento.

Controlador.

GRASP adicionales:

Fabricación Pura

Polimorfismo

Indirección

No hables con extraños

1.5.1 Patrón Bajo Acoplamiento

Este patrón estimula asignar una responsabilidad de forma tal que su empleo no incremente el acoplamiento, hasta el punto de que produzca los resultados negativos propios de un alto acoplamiento. Al mismo tiempo soporta el diseño de clases más independientes, reduce el impacto de los cambios, y estas son más reutilizables, que incrementan la oportunidad de una mayor productividad. Este patrón no puede considerarse en forma independiente a patrones como Experto y Alta Cohesión. Dentro de sus ventajas están:

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar.

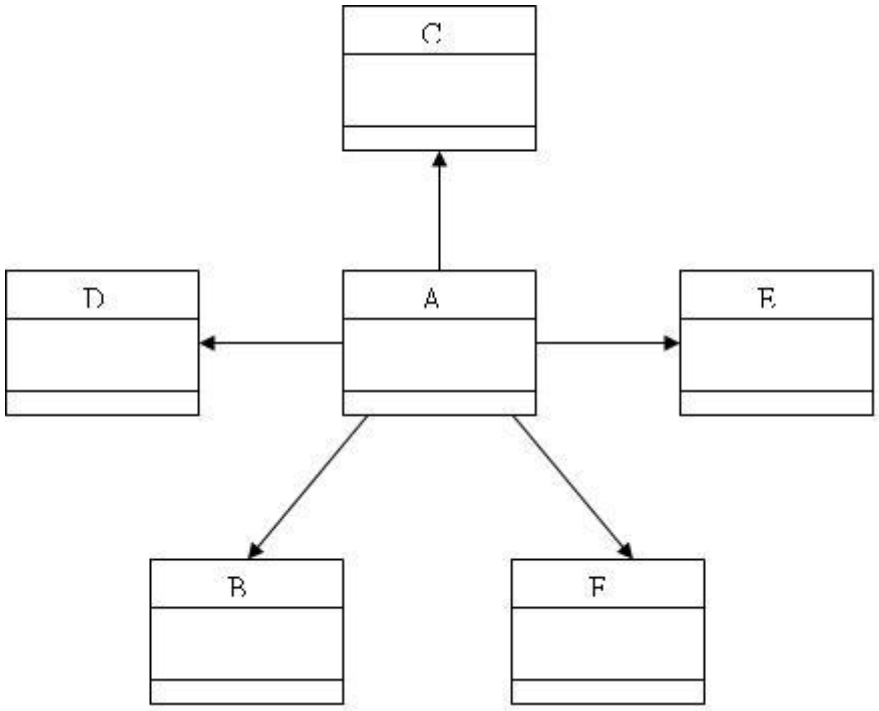

Figura 1 Patrón Bajo Acoplamiento.

1.5.2 Patrón Alta Cohesión.

Este patrón indica que una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Dentro de sus ventajas están:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo, se genera un bajo acoplamiento.
- Soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Es una de los patrones de asignación de responsabilidades o GRASP. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo. Esto presenta los siguientes problemas:

- Son difíciles de comprender.
- Difíciles de reutilizar.
- Difíciles de conservar.

1.5.3 Patrón Adapter (Adaptador).

La definición formal del adaptador es:

Convertir la interfaz de una clase en otra interfaz de cliente esperada.

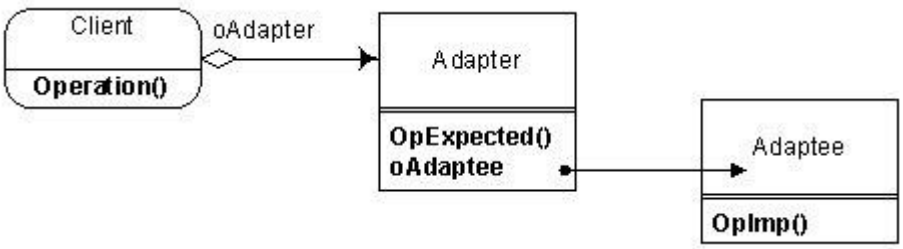

Figura 2 Patrón Adapter.

El patrón adaptador se aplica:

- Cuando se quiera utilizar una clase ya existente y su interfaz no se corresponda con la interfaz que se necesita.
- Se quiera definir una clase que se pueda utilizar para que pueda colaborar con otras clases cuyas interfaces no se conocen inicialmente.
- Para diseñar un visualizador genérico de estructuras compuestas.

Entre las principales ventajas e inconvenientes se encuentran:

- Una misma clase adaptadora puede adaptar una clase adaptada y a todas sus subclases.
- La adaptación no siempre consiste en cambiar el nombre y/o los parámetros de las operaciones, puede ser más compleja. (Erich Gamma)

1.5.4 Patrón Prototype (Prototipo).

Es un patrón de creación. El objetivo de este patrón es especificar los tipos de objetos a crear por medio de una instancia que hace de prototipo, creando nuevos objetos copiando dicha instancia. Todo se basa en clonar un prototipo dado.

Se debe usar este patrón cuando un sistema deba ser independiente de cómo se crean, se componen y se representan sus productos, y además:

- Cuando las clases a instanciar sean especificadas en tiempo de ejecución.
- Para evitar construir una jerarquía de clases de fábricas paralelas a la jerarquía de clases de producto.

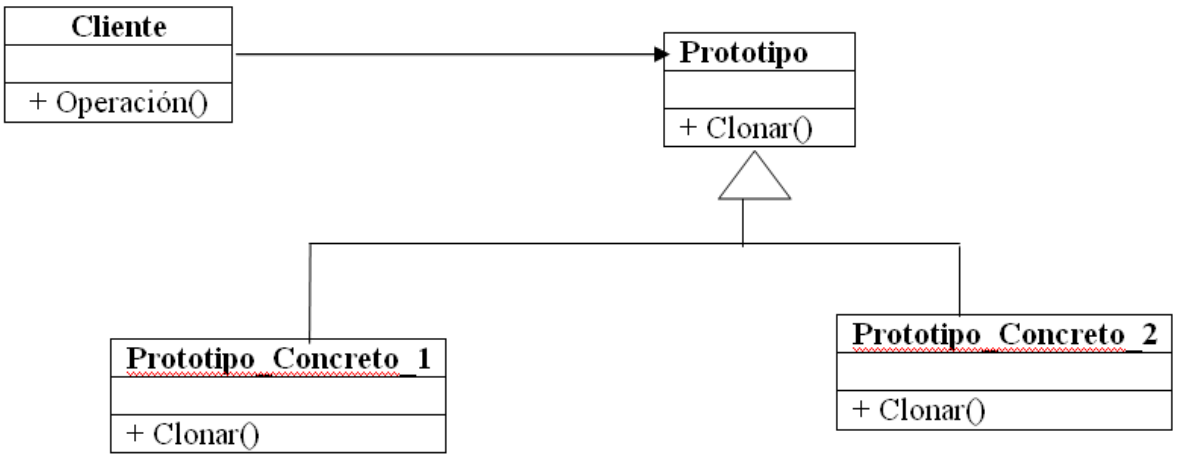

Figura 3 Patrón Prototype.

1.5.5 Patrón Singleton (Solitario).

La intención de este patrón es asegurarse de que una clase tiene una sola instancia y proporciona un punto de acceso global a ella. También es conocido como “Clases de Instancia Única” o “Instancia Única”. Se utiliza cuando se deba hacer una instancia a una clase y debe ser accesible a los clientes desde un punto de acceso conocido. Entre las principales ventajas e inconvenientes se tienen:

- Acceso controlado a la única instancia
- Espacio reducido, no hay variables globales.
- Puede adaptarse para permitir más de una instancia.

- Puede hacerse genérica mediante templetes. (Erich Gamma)

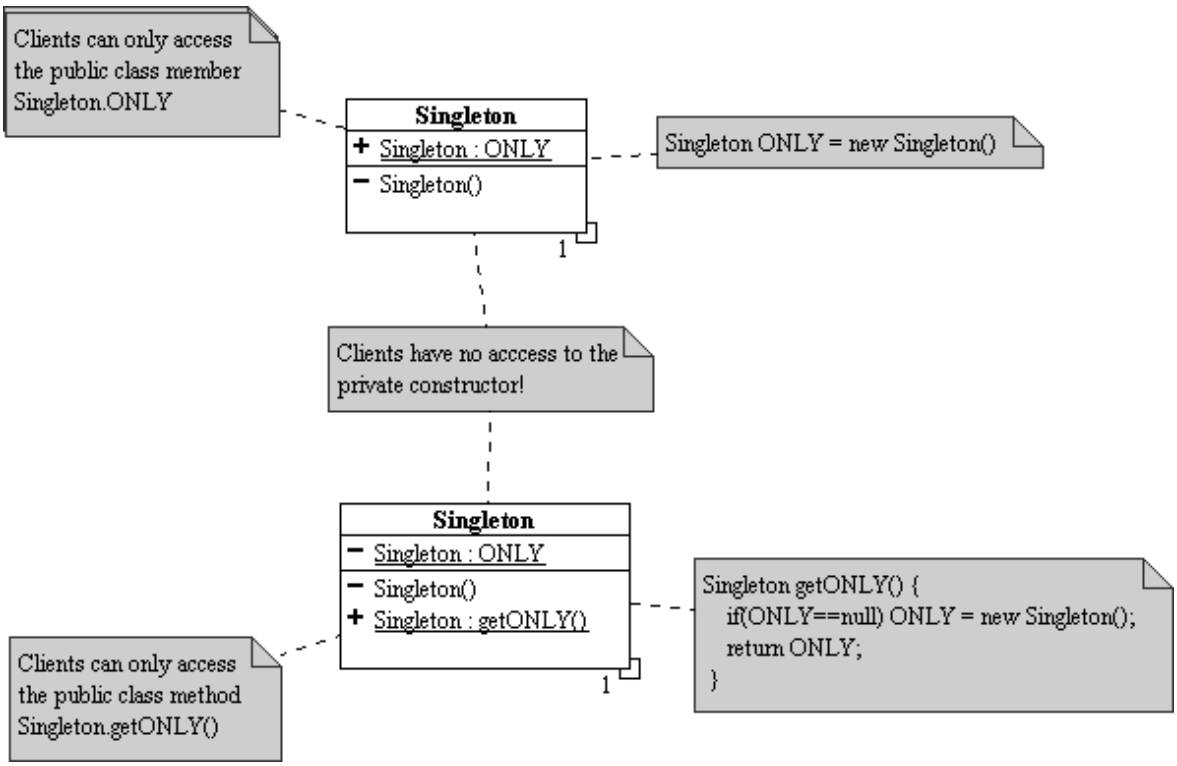

Figura 4 Patrón Singleton.

1.5.6 Patrón Builder (Constructor).

La intención de este patrón de diseño es separar la construcción de un objeto complejo de su representación. Los objetos que dependen de un algoritmo tendrán que cambiar cuando el algoritmo cambia; por lo tanto, los algoritmos que estén expuestos a dicho cambio deberían ser separados, permitiendo de esta manera reutilizar algoritmos para crear diferentes representaciones. El patrón Builder se usa cuando:

- El algoritmo para creación de un objeto complejo debe ser independiente de las partes que conforman el objeto y cómo están ensambladas.
- El proceso de construcción debe permitir diferentes representaciones del objeto que se construye.

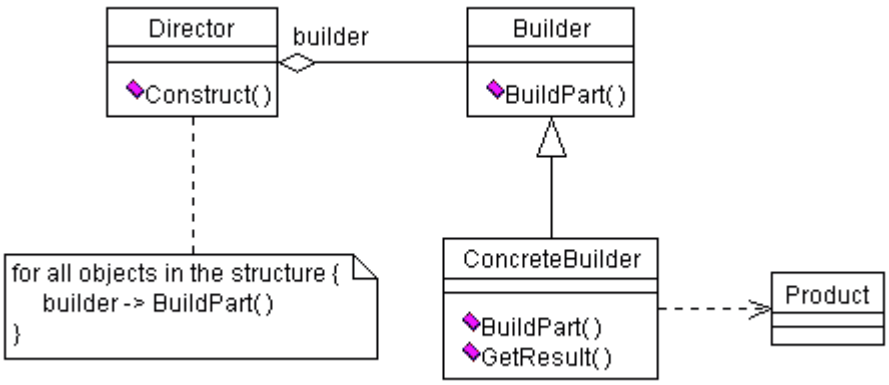

Figura 5 Patrón Builder.

1.5.7 Patrón Facade (Fachada).

Proporciona una interfaz unificada para un conjunto de interfaces de un sistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. La intensión de este patrón es proporcionar una interfaz simplificada para un grupo de subsistemas o un sistema complejo. Se utiliza cuando:

- Se quiera proporcionar una interfaz sencilla para un subsistema complejo.
- Se quiera desacoplar un subsistema de sus clientes y de otros subsistemas, haciéndolo más independiente y portable.
- Se quiera dividir los sistemas en niveles: las fachadas serían el punto de entrada a cada nivel.

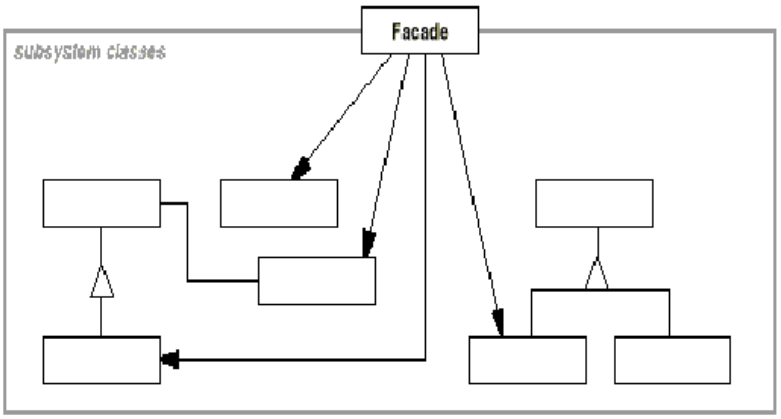

Figura 6 Patrón Facade.

1.5.8 Patrón Composite

La intensión de este patrón es componer objetos en estructura de árbol para representar jerarquías de tipo “partes-todo”. Permite tratar objetos individuales y composiciones de objetos de manera uniforme. Entre sus ventajas están:

Define jerarquías entre las clases.

Simplifica la interacción de los clientes.

Hace más fácil la inserción de nuevos hijos.

Hace el diseño más general.

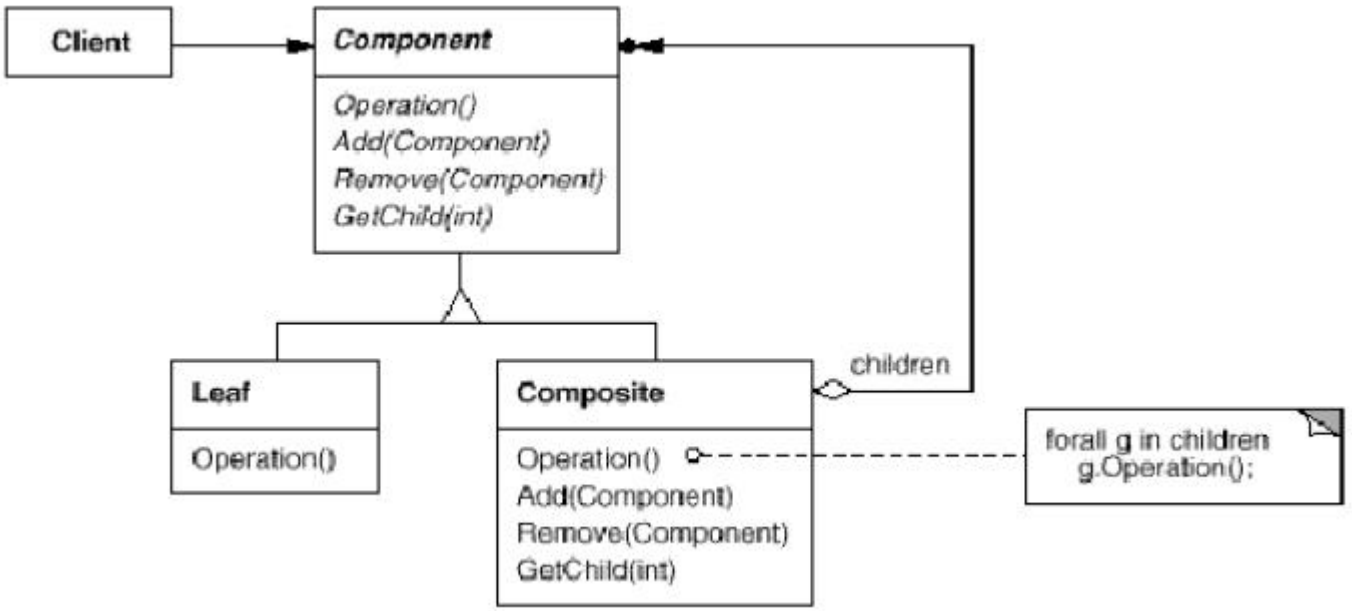

Figura 7 Patrón Composite.

1.5.9 Relación entre los niveles de Abstracción.

Con los estudios realizados de las diferentes definiciones de estilos arquitectónicos, patrones arquitectónicos y patrones de diseño, la figura que se muestra a continuación presenta la relación de abstracción existente entre estos conceptos. En ella se representa el planteamiento de Buschmann et al.

[8], que proponen el desarrollo de arquitecturas de software como un sistema de patrones, y distintos niveles de abstracción. (Ivar Jacobson, 2000)

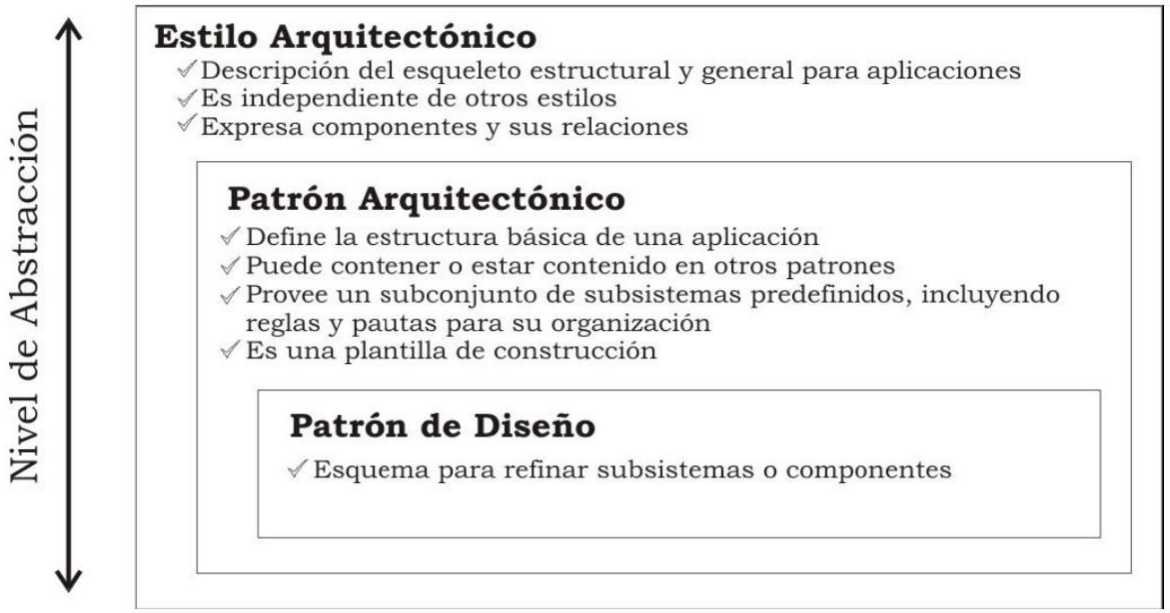

Figura 8 Relación de abstracción entre estilo arquitectónico, patrón arquitectónico y patrón de diseño.

Los estilos y patrones ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad. Asimismo la organización del sistema de software debe estar disponible para todos los involucrados en el desarrollo del sistema, ya que establece un mecanismo de comunicación entre los mismos. Estas descripciones pueden establecerse a través de las vistas arquitectónicas, las notaciones como UML (Lenguaje Unificado de Modelado) y los lenguajes de descripción arquitectónica.

1.6 Metodología RUP.

RUP es un proceso de desarrollo de software. Un proceso de desarrollo de software es un grupo de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. RUP además de un proceso es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, utiliza UML para crear los esquemas de un sistema, UML es una parte fundamental en RUP. El ciclo de RUP se define por ser iterativo e incremental, dirigidos por casos de uso y centrado en la arquitectura. (Ivar Jacobson, 2000)

1.7 Lenguaje Unificado de Modelado.

El **Lenguaje Unificado de Modelado (UML)** se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema de notación destinado a los sistemas de modelado que utilizan conceptos orientados a objetos. Proporciona a los desarrolladores un vocabulario que incluye tres categorías: elementos, diagramas y relaciones.

Los conceptos en UML son divididos en varias vistas que facilitan su comprensión, estas son: vista estática, vista de casos de uso, vista de máquina de estado, vista de actividades, vista de interacción y las vistas físicas. Las mismas son un subconjunto que modelan elementos que representan varios aspectos de un sistema. Los conceptos de cada vista son representados mediante una o dos clases de diagramas que proporcionan una notación visual.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema. (Ivar Jacobson, 2000)

1.8 Conclusiones parciales

En este capítulo se han expuesto conceptos relacionados con la arquitectura de software, desarrollo basado en componentes que son fundamentales para la comprensión de la presente investigación. Se abordaron temas relacionados con la arquitectura de software como son las ventajas y características de los principales estilos y patrones arquitectónicos. Se analizaron las características de un conjunto de tecnologías, herramientas y tendencias asociadas con la arquitectura para solucionar el problema científico que concierne la presente investigación. Se estudió la metodología de desarrollo del Proceso Unificado de Desarrollo de Software (RUP) y como lenguaje de modelado el UML. De esta manera se proporcionan los conocimientos teóricos necesarios para el entendimiento del presente trabajo.

CAPÍTULO 2: DESCRIPCIÓN DE LA ARQUITECTURA

En este capítulo se tiene como objetivo lograr una mejor visión del sistema, organizar el desarrollo, posibilitando esto que el desarrollo del sistema sea de forma eficiente. Aquí se obtienen artefactos fundamentales como la Línea Base de la Arquitectura además se describe la solución arquitectónica del sistema, además otros artefactos definidos por la metodología RUP como son las vistas arquitectónicas.

2.1 Selección de las herramientas.

2.1.1 ARToolkit.

ARToolkit es un conjunto de bibliotecas para C/C++ que sirve para la creación de aplicaciones de Realidad Aumentada. No es directamente un software de creación como Flash, Dreamweaver o Director. Según el criterio de algunos autores es el principal software de Realidad Aumentada, y el primero en haber sido desarrollado y hacerse disponible.

Utiliza las capacidades de seguimiento de video, a través de técnicas de visión por computadoras, con el fin de calcular, en tiempo real, la posición de la cámara y la orientación relativa a la posición de los marcadores físicos. Una vez que la posición de la cámara real se conoce, la cámara virtual se puede colocar en correspondencia y los modelos 3D son superpuestos exactamente sobre el marcador real. (Aumentada)

Así ARToolkit resuelve dos de los principales problemas en la Realidad Aumentada, el seguimiento de objetos y la interacción con el objeto virtual. También proporciona una serie de ejemplos y utilidades de gran ayuda al programador que quiera realizar este tipo de aplicaciones.

Disponibilidad y distribución.

ARToolkit está siendo continuamente desarrollado por el Dr. Hirokazu Kato de la Universidad de Osaka, en Japón y es apoyado por el Human Interface Technology Laboratory de la Universidad de Washington y el Human Interface Technology Laboratory de la Universidad de Canterbury en Nueva Zelandia.

Periódicamente se lanzan nuevas versiones que se distribuyen a través de la página Web de HITLab (http://www.hitl.washington.edu/artoolkit). Su distribución es gratuita, y se entrega bajo la licencia GNU General Public License (http://www.gnu.org/licenses/gpl.html), lo que permite su uso comercial.

Pre-requerimientos y dependencias de ARToolkit en el entorno Windows.

ARToolkit puede compilase en los entornos Windows, Linux, SGI, Irix y MacOS. Si bien las funciones que ofrece en las distintas plataformas son las mismas, la instalación del programa varía en cada una de ellas.

El equipo, sistema operativo y plataforma deben satisfacer algunos requisitos mínimos básicos. El hardware utilizado debe ser capaz de recibir un input de video en vivo y tener suficiente capacidad de procesamiento disponible para cumplir con las tareas de proceso del video y display final. Como estas operaciones ocurren en tiempo real, es necesario un procesador capaz de manejarlo. De lo contrario, surgirían problemas como dropped frames y bajos frames por segundos en el aumento.

El software posee también varias dependencias. Estas dependencias son softwares adicionales que de forma accesoria cumplan funciones específicas en pro de ARToolkit para que éste pueda desarrollar su función final.

En esta investigación se decidió utilizar esta herramienta gracias a la flexibilidad, facilidades legales y su posición como el estándar establecido en cuanto a aplicaciones de Realidad Aumentada, en la que presenta muy buena aceptación. La mayoría de los softwares mencionados en este capítulo se basan en dicha biblioteca.

2.1.2 Visual Paradigm

Visual Paradigm es una de las herramientas CASE, fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Permite graficar los diferentes diagramas de UML, revertir y generar código fuente para Java, C++, PHP, DotNet Exe/dll, XML, XML Schema, Python y Corba IDL. (Paradigm)

Visual Paradigm ofrece soporte para Rational Rose, integración con Microsoft Visio. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto. Una de sus principales ventajas es que incorpora el soporte para trabajo en equipo, permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros. (Paradigm)

Beneficios:

- Soporte para toda la notación UML.
- Capacidades de ingeniería directa (versión profesional) e inversa.

- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.
- Se llegó a la conclusión de que Visual Paradigm es el más indicado para desarrollar el sistema que se desea implementar, además la Universidad de las Ciencias Informáticas cuenta con una licencia para su uso y funciona perfectamente sobre la plataforma GNU Linux. (Paradigm)

2.1.3 Framework de Desarrollo: QT-Designer.

Qt es un framework de desarrollo multiplataforma con una biblioteca de más de 400 clases, las cuales encapsulan toda la infraestructura necesaria para desarrollar aplicaciones robustas. Posee una utilidad para la rápida construcción de interfaz gráfica de usuario con apariencia y aspectos nativos de las plataformas soportadas (QT-Designer). Facilita el flujo de trabajo de internacionalización mediante QT Linguist. Está separado en 13 módulos y la API que brinda incluye funciones para conectividad a bases de datos, programación en red, integración con gráficos 3D, desarrollo multi-hilos y lectura/escritura de ficheros XML. (QT)

Introduce una alternativa innovadora para comunicación entre objetos, las llamadas señales y ranuras (signals y slots), que reemplazan la antigua e insegura técnica callback. Provee un modelo de eventos convencional para manipular los eventos del mouse, teclas presionadas y demás entradas del usuario. (QT)

Características:

- Características generales: El diseño modular de la biblioteca Qt proporciona un rico conjunto de aplicaciones que ofrecen todas las funcionalidades necesarias para la construcción de potentes interfaces gráficas de usuario. Qt es una biblioteca fácil de aprender y usar. Permite crear código altamente legible y fácil de mantener. (QT)
- Entorno de desarrollo: Qt cuenta con entorno desarrollo integrado (IDE, por sus siglas en inglés) y permite el diseño de las ventanas a través de una GUI, entre otras funcionalidades. (QT)
- Multi-plataforma: Qt está disponible para las plataformas Mac OSX, Windows, Linux/X11, Windows CE y S60. (QT)

- Lenguajes de programación: Qt proporciona una intuitiva interfaz de clases para el desarrollo de aplicaciones en C++. Incluso va un poco más allá del lenguaje C++ en lo referido a la comunicación entre objetos y la flexibilidad para el desarrollo de interfaces gráficas de usuario agregando un poderoso mecanismo de comunicación entre objetos. (QT)

2.1.4 OsgART.

OsgART es una biblioteca de C++ que ofrece la posibilidad de crear aplicaciones de Realidad Aumentada mediante la combinación de ARToolKit y OpenSceneGraph. Dispone de tres funciones principales: (Aumentada)

- Alto nivel de integración de entrada de video (video objeto, shaders).
- Soporte de múltiples marcadores basados en características físicas o tecnologías de seguimiento (marcador basado en múltiples rastreadores)
- Registro fotométrico, basado en técnicas para el renderizado de sombras y oclusión.

Características:

- Soporta el renderizado de sombras.
- Soporta varias entradas de video.

2.1.5 Motor gráfico: Open Scene Graph.

OpenSceneGraph (OSG) es un motor gráfico de código abierto, utilizado para el desarrollo de aplicaciones gráficas de alto rendimiento tales como, simuladores de vuelo, juegos, aplicaciones de realidad virtual y visualización científica. Esta herramienta está basada en el concepto de grafos de escena, provee una plataforma orientada a objetos que utiliza OpenGL como API gráfica. OSG está disponible tanto para uso comercial como no comercial. (OSG)

Características:

- Rendimiento: Soporta view-frustum culling, occlusion culling, small feature culling, LOD, vertex arrays, vertex buffer objects, OpenGL Shader Language y display lists. Todo esto hace de OSG una de los motores gráficos de más alto rendimiento disponible.

- Productividad: El núcleo de OSG encapsula la mayoría de las funcionalidades de OpenGL incluyendo las últimas extensiones de este, provee optimización de la visualización, y un conjunto de bibliotecas adicionales las cuales hacen posible desarrollar aplicaciones gráficas de alto rendimiento muy rápidamente. Combinando las experiencias de otros motores gráficos con modernos métodos de ingeniería de software como patrones de diseño, se ha logrado el diseño de una biblioteca robusta y extensible.

- Extensiones Soportadas: Para la lectura y escritura de archivos la biblioteca (osgDB) proporciona una amplia variedad de formatos por medio de un mecanismo extensible de plugins dinámicos. Actualmente incluye cincuenta y cinco plugins para cargar formatos 3D y formatos de imágenes.

- Portabilidad: El núcleo de OSG estás diseño para tener una dependencia mínima de cualquier plataforma. Esto ha permitido que OSG fuese rápidamente portado a un amplio rango de plataformas. Originalmente fue desarrollado en IRIX, luego portado a Linux, Windows, FreeBSD, Mac OSX, Solaris, HP-UX, AIX y hasta PlayStation2.

- Soporte multi-lenguaje: OSG está disponible además en los lenguajes Java, Lua y Pyhton. (OSG)

2.1.6 Motor gráfico OGRE.

Está diseñado desde un comienzo con la idea de la orientación a objetos, por lo que su interfaz es clara, intuitiva y fácil de emplear. Es un proyecto bajo licencia LGPL por lo tanto es gratis su uso. No posee nada que envidiarle a los motores gráficos más nuevos disponibles en el mercado. Entre sus características se encuentra el soporte para VS 3.0 y exportadores para las herramientas de modelación tridimensional más utilizadas en las empresas desarrolladoras de videojuegos. Esto significa que con OGRE se logra crear juegos o cualquier prototipo de aplicación que trabaje con gráficos tridimensionales que no tengan nada que envidiarle a lo más moderno de la industria de los videojuegos. OGRE está orientado al crecimiento de la escena, y diseñado con el objetivo principal de permitir a los programadores producir aplicaciones utilizando gráfico en 3D acelerados por hardware. Brinda como beneficio fundamental su diseño coherente y la documentación especificada y consistente que viene con el motor.

2.1.7 Formato para el almacenamiento de datos: XML.

XML (eXtensible Markup Language o lenguaje de marcas extensible) es un metalenguaje extensible desarrollado por el World Wide Web Consortium (W3C). XML es un formato basado en texto, fue diseñado específicamente para almacenar y transmitir datos. Su principal novedad consiste en compartir los datos

con los que trabajan todos los niveles, por todas las aplicaciones y soportes. Permitiendo compartir la información de una manera segura, fiable y fácil. (Romero, 2008)

Principales objetivos:

- Los documentos XML deben ser legibles y claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser de fácil creación.
- La concisión en las marcas XML es de mínima importancia.

Características:

Entre las características de XML que han logrado que se convierta en un formato universal, se encuentran las siguientes: (Romero, 2008)

- Es una arquitectura más abierta y extensible. No se necesita versiones para que pueda funcionar en futuros navegadores.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje permite agrupar una amplia variedad de aplicaciones, desde páginas web hasta bases de datos.
- Gestión y manipulación de los datos desde el propio cliente Web.
- Se permitirá un comportamiento más estable de las aplicaciones Web.

Puede exportar a otros formatos de publicación. El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

Ventajas:

- Entre las principales ventajas que ofrece XML se pueden citar las siguientes:
- Estandarización de la Información.
- Integración de aplicaciones.

- Portabilidad de Información.
- Mejora el acceso a la información.

2.1.8 Lenguaje de Programación C++.

El C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C.

Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos.

Características.

- Programación orientada a objetos: La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real. Permite la reutilización del código de una manera más lógica y productiva para el uso de plantillas.
- Portabilidad: Un código escrito en ANSI C++ puede ser compilado en casi todo tipo de ordenadores y sistemas operativos sin hacer apenas cambios, siempre que exista el compilador.
- Brevedad: El código escrito en C++ es muy corto en comparación con otros lenguajes, sobretodo porque en este lenguaje es preferible el uso de caracteres especiales que las "palabras clave".
- Programación modular: Un cuerpo de aplicación en C++ puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Esta característica permite unir código en C++ con código en otros lenguajes de programación como Ensamblador o el propio C.
- Velocidad: El código resultante de una compilación en C++ es muy eficiente, gracias a su capacidad de actuar como lenguaje de alto y bajo nivel.
- Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:
- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Identificación de tipos en tiempo de ejecución (RTTI).

- C++ es uno de los lenguajes más potentes, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje.

2.2 Línea base de la arquitectura.

La Línea Base de la Arquitectura contiene elementos de gran importancia para lograr la máxima abstracción en el diseño arquitectónico de la aplicación. En la misma se exponen los estilos arquitectónicos de la aplicación, así como los principales elementos de la arquitectura, se describen los principales patrones de arquitectura utilizados, las tecnologías y herramientas de software que se utilizarán en el sistema a desarrollar.

2.2.1 Propósito.

El propósito de la línea base es proporcionar la información para estructurar el sistema desde el más alto nivel de abstracción, se define la estructura del sistema en cuanto a los elementos, la configuración y sus restricciones.

Los usuarios de la línea base de la arquitectura son:

- Equipo de arquitectos del proyecto, guía la toma de decisiones que tengan que ver con la arquitectura, son los encargados del mantenimiento y refinamiento de la línea base.
- Los miembros del equipo de desarrollo encargados de realizar la aplicación, la utilizan para guiar la implementación del sistema.
- Los clientes tienen en ella una garantía de la calidad y el conocimiento sobre la tecnología en que está desarrollada la solución.

2.2.2 Alcance.

Se describe la estructura de la aplicación en su más alto nivel de abstracción. Describe detalladamente al organigrama de la arquitectura encarnada en los estilos arquitectónicos que se utilizarán, se propone la utilización de un conjunto de patrones que van a resolver futuros problemas que se van a presentar a lo largo del desarrollo del sistema. Se proponen las principales tecnologías y herramientas que soportan los estilos y patrones especificados y cumplen con las restricciones del sistema.

2.3 Arquitectura basada en componentes.

El desarrollo de software basado en componentes puede verse como una extensión natural de la programación orientada a objetos dentro del ámbito de los sistemas abiertos y distribuidos, se basa en el uso de los componentes de software como entidades básicas del modelo. Esto posee algunas ventajas las cuales sirven de apoyo para dar nuestra propuesta para definir la arquitectura del proyecto.

- **Reutilización del software:** Permite reutilizar componentes ya definidos.
- **Simplifica el mantenimiento del sistema:** Al existir un débil acoplamiento entre componentes, el desarrollador puede actualizar y/o agregar componentes sin afectar otras partes del sistema
- **Mayor calidad:** Dado que un componente puede ser construido y luego mejorado continuamente, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

2.4 Modelo de Dominio.

El Proceso Unificado de Desarrollo está basado en componentes y tiene como características principales: guiado por casos de uso, centrado en la arquitectura e iterativo e incremental. Utiliza el lenguaje UML (lenguaje unificado de modelado) para preparar todos los esquemas de un sistema de software. De hecho, UML es una parte esencial de RUP, ya que sus desarrollos fueron paralelos. En este caso debido a la poca estructuración de los procesos de negocio se plantea un modelo de dominio para lograr una mejor compresión de los conceptos del sistema. Para esto se realiza la descripción del modelo del dominio a través de un diagrama de clases UML, en el cual se definen las principales clases conceptuales que intervienen en el sistema.

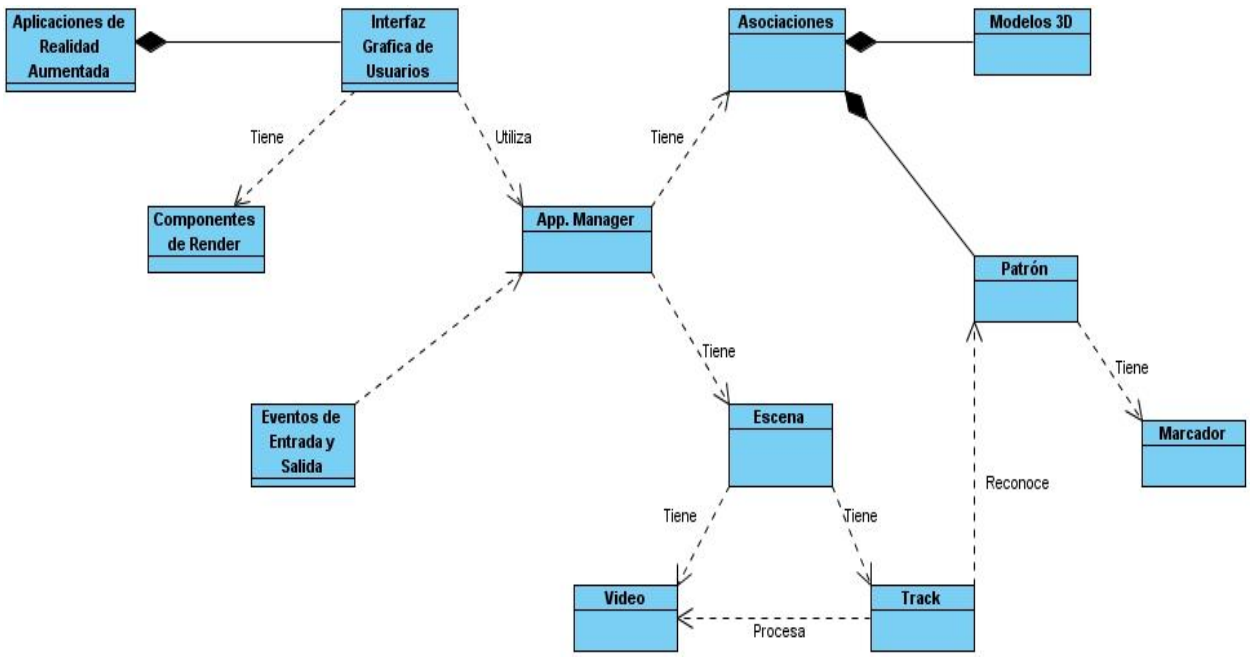

Figura 9 Modelo de Dominio.

A continuación se explican los términos que forman parte del modelo de dominio para su mejor comprensión.

Aplicaciones de Realidad Aumentada:

Interfaz Gráfica de Usuario: Procesa la entrada y salida de datos para la aplicación, debe estar compuesto por campos editables o menús. Esta tiene componentes de render y utiliza el manejador de aplicaciones.

Asociaciones: Relaciona los modelos 3D con los patrones.

Patrón: Mapa de bits generado a partir de la región que ocupa un marcador determinado en cada fotograma del video capturado por la cámara y son basados en marcadores.

Componentes de Render: Permite acoplarse al contexto gráfico, entre sus principales funcionalidades se encuentran la de dibujado y los ciclos inactivos.

App Manager (Manejador de Aplicaciones): Este es un concepto abstracto el cual expresa que es el encargado de manipular otros conceptos. Este tiene escena y asociaciones.

Modelos 3D: Modelos virtuales de objetos reales en 3D.

Marcadores: Figura impresa en un papel en blanco y negro.

Video: Está compuesto por un conjunto de imágenes de la escena que se suceden. Este es generado por la cámara.

Tracker: Permite añadir y detectar objetos de seguimiento, realiza el reconocimiento y los registros 3D.

Eventos de Entrada y Salida: Capturar la entrada y salida que se originen a través de los dispositivos de entrada y salida como mouse, teclado y otros dispositivos que se utilicen en la aplicación.

2.5 Estilo arquitectónico propuesto.

Durante el estudio realizado se escogió para el desarrollo del producto el patrón arquitectónico MVC que forma parte del estilo arquitectónico de llamada y retorno. El patrón MVC cumple con el patrón de diseño alta cohesión. Cada una de las partes son independientes, la comunicación entre ellas es mediante interfaces, que se abstraen en sus estructuras internas. Esto permite desarrollar el modelo, la vista y la controladora de forma independiente sin afectar a las demás.

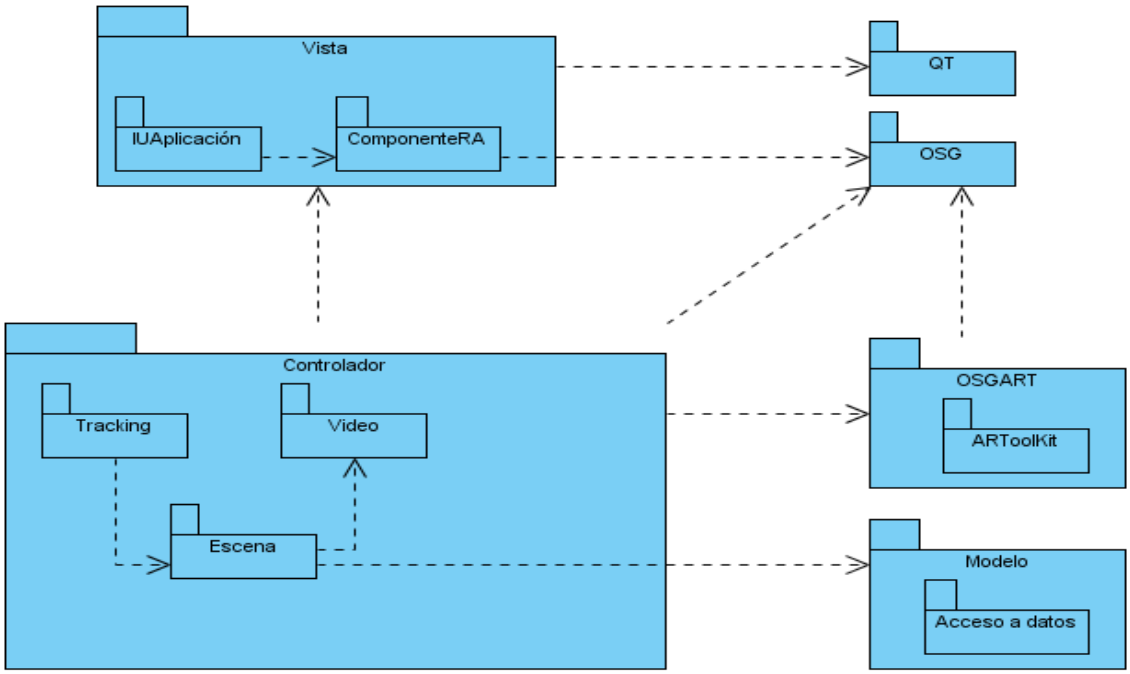

Figura 10 Patrón MVC, Estilo arquitectónico propuesto.

2.5 Requisitos Funcionales.

RF1- Reproducir video de la escena.

RF1.1- Reproducir Video desde Cámara (resolución de la cámara, velocidad de captura y formato de video).

RF1.2- Reproducir Archivo desde Archivo (dirección y formato del video).

RF1.3- Reproducir desde dirección remota (URL o dirección remota donde se encuentra la fuente de video (desde servidor de streamming)).

RF2- Cargar recursos para representar en la escena.

RF2.1- Cargar Modelos 3D.

RF2.2 -Cargar Archivos.

RF3- Hacer registro 3D.

RF3.1-Trackers por visión por computadoras.

RF4- Crear asociaciones de recursos (recursos - datos de registro).

RF5-Seleccionar recurso virtual en escena aumentada.

RF5.1- Dibujar Escena Aumentada.

RF6- Interacción con el usuario.

RF7- Persistencia de datos.

2.6 Requisitos No Funcionales.

Requisitos de apariencia.

- El sistema debe contener una interfaz sencilla e intuitiva, que brinde facilidades y sea agradable al usuario.
- La aplicación debe utilizar como idioma principal el español, excepto aquellas palabras técnicas que no puedan ser traducidas.
- Utilizar botones que expresen su función, ya sea que se intuya o expresados con texto.

Requisitos de usabilidad.

- La aplicación deberá poseer una interfaz y navegación acorde y funcional, tanto para usuarios expertos, como para los que no tienen conocimientos profundos de Informática.

Requisitos de software.

- Las computadoras que utilizarán el software deben tener instalado: Windows 2000 NT, Windows XP Professional.

Requisitos de hardware.

- Las computadoras que utilizarán el software a desarrollar deberán tener 1 GB de Memoria RAM como mínimo.

Requisitos de diseño e implementación.

- El contenido se cargará desde archivos XML.

Requisitos de Rendimiento.

- La herramienta propuesta debe ser rápida.
- Haga uso óptimo de la memoria.

2.7 Vistas arquitectónicas.

Siguiendo la metodología RUP tal y como se expuso en el Capítulo 1, a continuación se desarrolla la descripción de la arquitectura a través de las 4+1 vistas propuestas por Kruchten, es importante destacar que una de las vistas propuestas por este modelo, la Vista de Procesos, no es necesaria ya que el sistema no maneja hilos de procesos concurrentes aunque las biblioteca de OSGART sí utiliza esta programación.

2.7.1 Vista de Casos de Usos

La descripción textual de los Casos de Uso se puede observar en el **Anexo 1.**

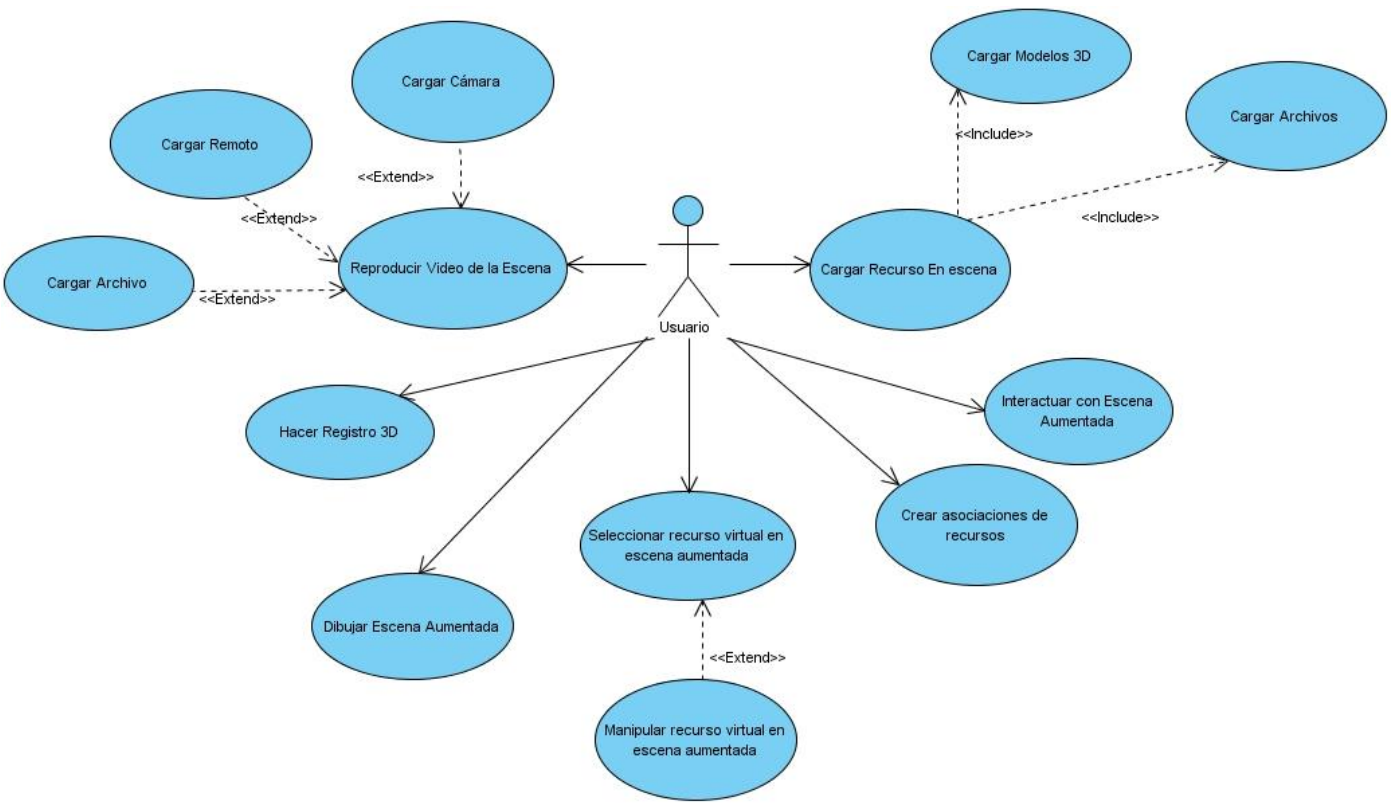

Figura 11 Vista de Casos de Usos.

2.7.2 Vista lógica

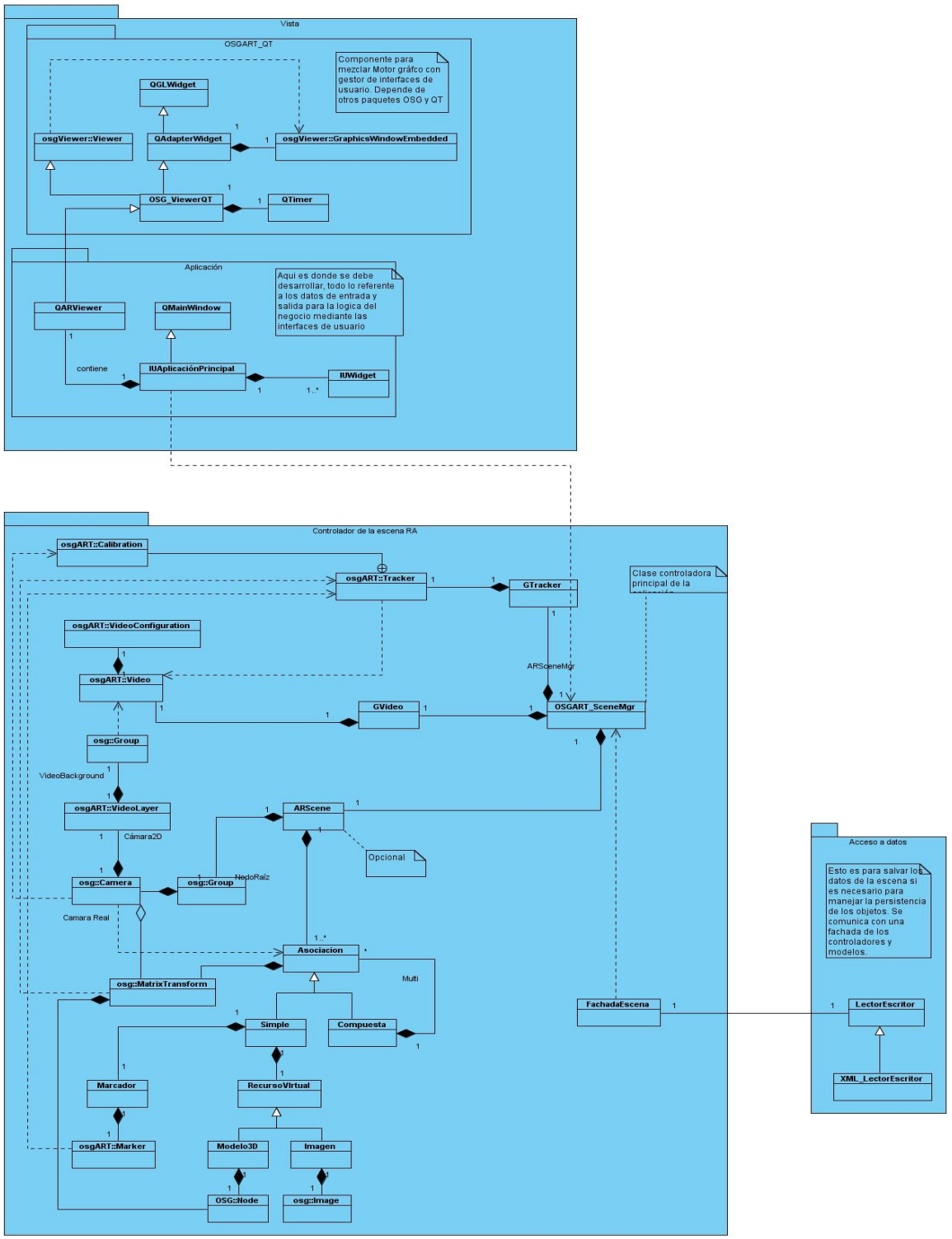

Figura 12 Vista Lógica.

2.7.3 Vista de implementación.

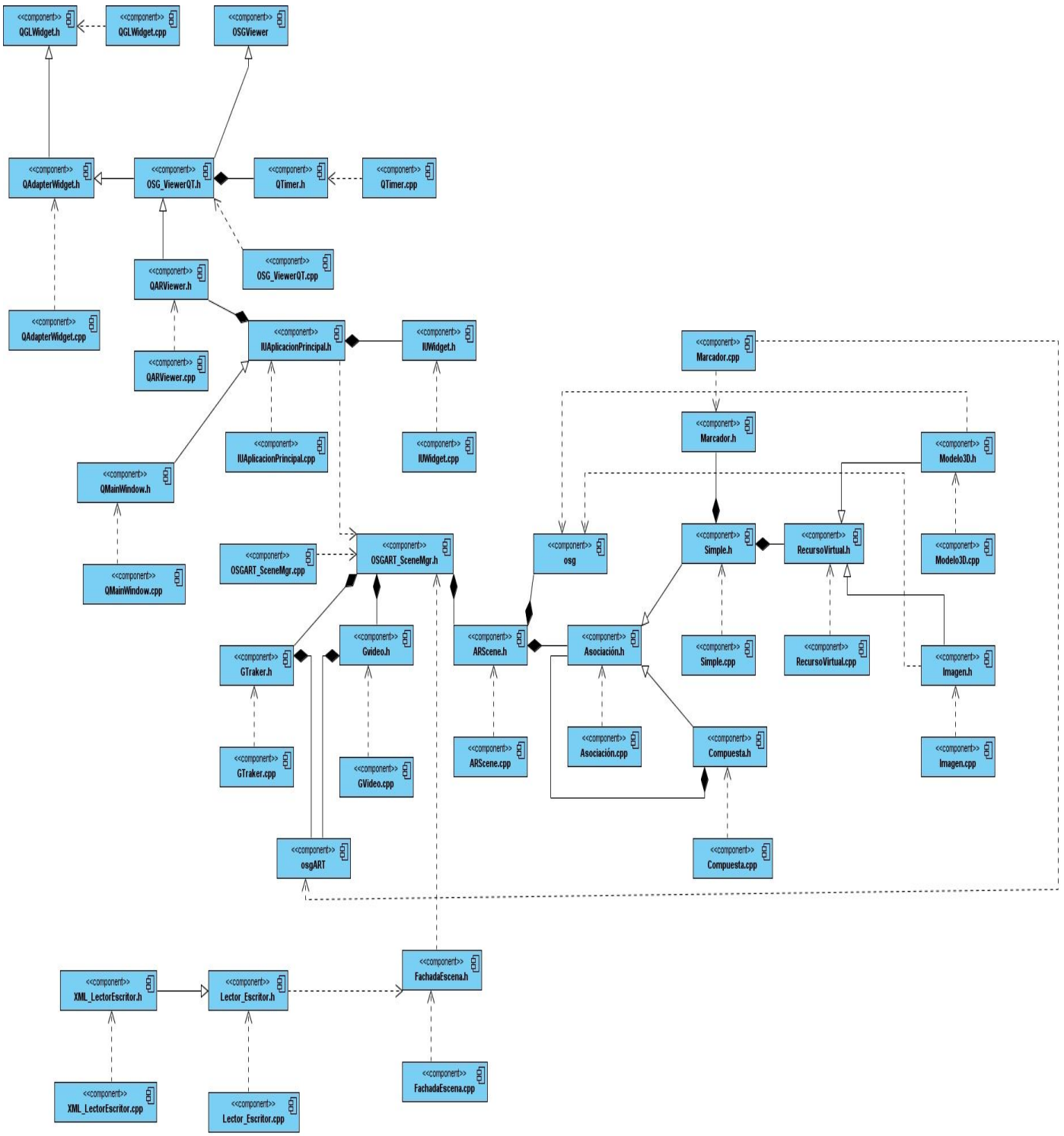

Figura 13 Diagrama de Implementación.

2.7.4 Vista de despliegue.

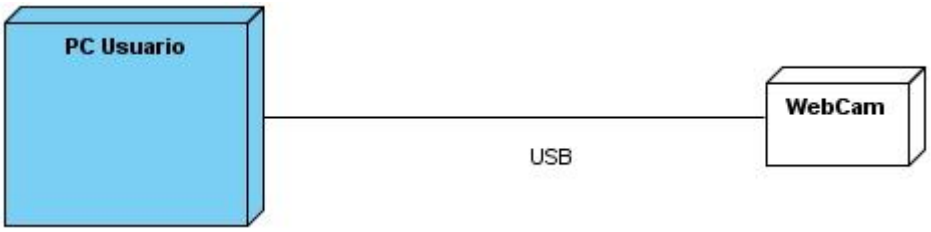

Figura 14 Diagrama de Despliegue.

PC Usuario: Tiene que tener instalado el software que soporte esta arquitectura para permitir utilizar las aplicaciones de realidad aumentada y además debe de contar con uno o más puertos de USB para conectar la webcam.

Webcam: Permite tomar la imagen que serán procesada en la escena aumentada.

2.8 Patrones propuestos.

Los patrones que se proponen son:

- **Bajo acoplamiento.**
- **Alta cohesión.**

De estos patrones no se ponen ejemplos porque son principios que se deben tener en cuenta a la hora de diseñar la mayoría de las clases.

- **Adapter.**

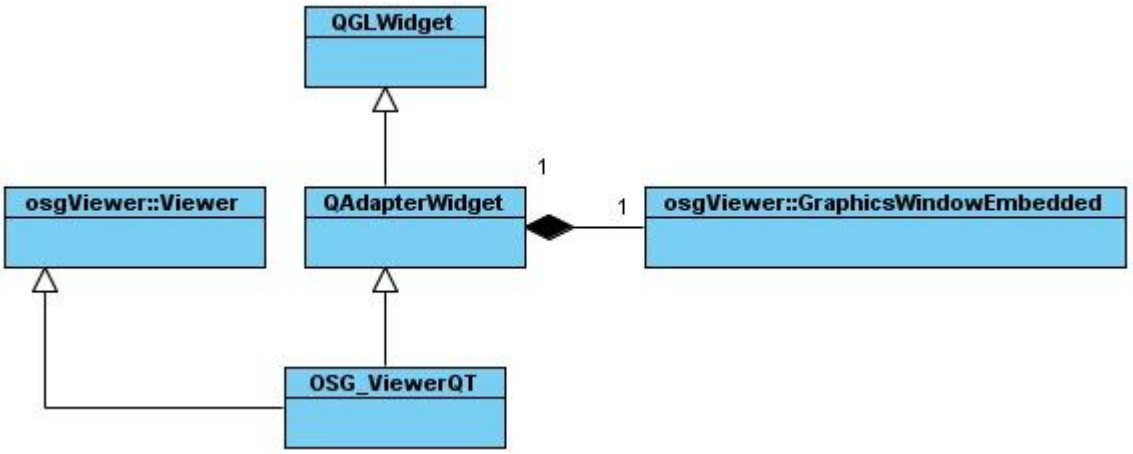

Figura 15 Patrón Adapter en la arquitectura propuesta.

- **Singleton.**

Figura 16 Patrón Singleton en la arquitectura propuesta.

- **Facade.**

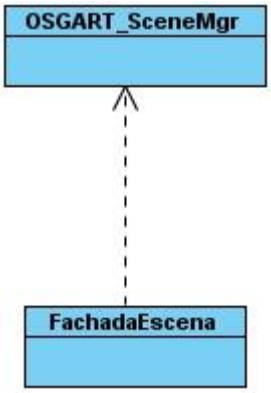

Figura 17 Patrón Facade en la arquitectura propuesta.

Composite.

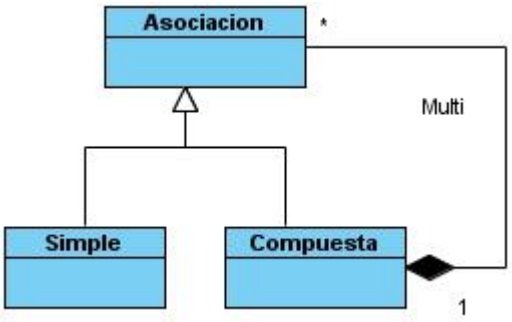

Figura 18 Patrón Composite en la arquitectura propuesta.

- **Builder.**

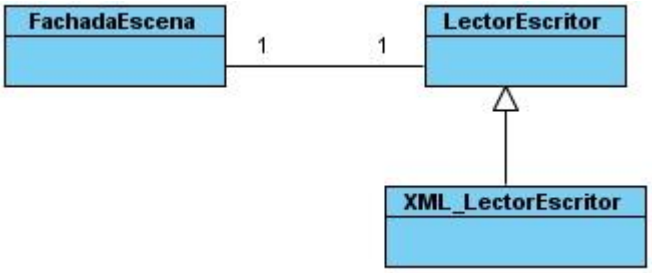

Figura 19 Patrón Builder en la arquitectura propuesta.

2.11 Conclusiones parciales.

En este capítulo se realiza la propuesta de la arquitectura del proyecto ARLab, definiendo las herramientas para modelado y desarrollo, la estructura del equipo de desarrollo, los módulos que conforman al proyecto, las vistas arquitectónicas, los requisitos no funcionales y funcionales. A partir de los resultados obtenidos ya está disponible todo para la implementación.

CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA.

La arquitectura es un artefacto definitivo para la calidad del software que se desarrolla. Durante la evaluación de la misma se puede conocer los diferentes riesgos asociados con el desarrollo del sistema y como la arquitectura es un producto temprano son muchos los errores que se pueden corregir durante el flujo de requerimiento o diseño siendo estos menos costosos que si tuvieran que aclararse en implementación o prueba. En este capítulo se analizan los resultados obtenidos a través de los atributos de calidad que propone el modelo de calidad ISO/IEC 9126.

3.1 Evaluación de la Arquitectura.

La evaluación tiene como objetivo saber si la arquitectura puede habilitar los requerimientos no funcionales y los atributos de calidad para asegurar que el sistema cumpla con las necesidades de los clientes, por esto es muy importante evaluarla.

El primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. (Kazman, 2001)

Evaluar la arquitectura de un software sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber qué tan adecuada es la arquitectura diseñada para el sistema.

La evaluación de la arquitectura no determina la calidad de esta, solo identifica dónde están los riesgos, las fortalezas y las debilidades. Se pueden tomar algunas decisiones definiendo si se puede continuar el proyecto dada las debilidades identificadas, si hay que reforzar la arquitectura o comenzarla de nuevo. Es importante saber en qué etapa evaluar la arquitectura para poder determinar con mayor exactitud los posibles riesgos. (Bosch, 2000)

3.2 Etapas en la que se evalúa la arquitectura.

Existen dos períodos para evaluar la arquitectura por decirlo de alguna manera, usted puede decidirse por la que más factible le sea, de acuerdo con el proyecto que se esté desarrollando:

Evaluación temprana: No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

Evaluación tardía: Cuando ésta se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado.

3.3 Atributos de Calidad.

La AS se rige por las cualidades para su evaluación, éstos son los atributos de calidad, aunque a veces son demasiados inconcretos. Estos se clasifican en dos tipos: observables en vías de ejecución y no observables en vías de ejecución.

Observables mediante la ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.

No observables en la ejecución: aquellos atributos que se establecen durante el desarrollo del sistema.

3.4 Modelo de calidad ISO/IEC 9126.

Diversos son los modelos de calidad a través de los años, los mismos marcan una pauta importante a la hora de determinar el nivel de complejidad de un determinado producto de software. Fueron establecidos de acuerdo con el impacto que tienen los atributos de calidad, en lo que a evaluación de la arquitectura se refiere.

Variadas han sido las propuestas desde 1970 y durante los años siguientes hasta la actualidad, donde se encuentra el modelo ISO/IEC 9126, en el cual estará centrado este capítulo en cuanto a modelos de evaluación, debido a que el mismo se basa en los atributos de la calidad que tienen un vínculo directo con la arquitectura.

Según lo planteado en la ISO 9126, la calidad de un software debe realizarse sobre la base de sus atributos, tanto internos como externos. Es decir, la calidad interna, que trata la estructuración de las propiedades del software, influye directamente sobre la externa, o sea, cualidades observables sin conocer como está construido el producto.

Estos atributos se expresan en un conjunto de características bien definidas en la norma ya mencionada:

Funcionalidad:

- Educación: capacidad del producto de software para proporcionar un conjunto de funciones para tareas específicas.

- Interoperabilidad: capacidad del producto para interactuar con más sistemas.
- Seguridad de acceso: capacidad para proteger la información y los datos.

Fiabilidad:

- Madurez: capacidad del producto para evitar fallas.
- Tolerancia a fallos: capacidad de mantener un nivel de prestaciones en caso de fallos.
- Capacidad de recuperación: capacidad del software para restablecer las prestaciones y recuperar los datos dañados.

Usabilidad:

- Capacidad para ser entendido: capacidad que permite que el usuario pueda entender si el producto es el adecuado.
- Capacidad para ser aprendido: capacidad que permite al usuario operar y controlar el software.
- Capacidad de atracción: capacidad del producto de ser atractivo al usuario.

Eficiencia:

- Comportamiento temporal: capacidad del producto para proporcionar tiempos de respuestas bajo condiciones determinadas.
- Utilización de recursos: capacidad del software para utilizar la cantidad y tipo de recurso más adecuado.

Mantenibilidad:

- Capacidad para ser analizado: capacidad del producto para diagnosticar deficiencias o fallos.
- Capacidad de ser cambiado: capacidad de permitir hacerle modificaciones.
- Estabilidad: capacidad del producto para evitar efectos inesperados.
- Capacidad para ser probado: capacidad que debe permitir que el software modificado sea válido.

Portabilidad

- Adaptabilidad: capacidad del producto para adaptarse a diferentes entornos.

- Instalabilidad: capacidad del producto de ser instalado en un entorno determinado.
- Coexistencia: capacidad que tiene el producto de poder coexistir con otros software en un entorno determinado.
- Capacidad para remplazar: capacidad que tiene el producto para remplazar a otro software.

3.6 Técnicas de evaluación.

Las técnicas utilizadas para la evaluación de atributos de calidad demandan grandes esfuerzos por parte del ingeniero de software para crear especificaciones de predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema.

Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño existen diferentes técnicas de evaluación: basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia.

3.6.1 Evaluación basada en escenario.

Un escenario es una breve descripción de la interacción de algunos de los involucrados en el desarrollo del sistema con este. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. El contexto describe que sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, como debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. Los escenarios proveen en vehículo que permite concretar y entender atributos de calidad.

Entre sus principales ventajas están:

1. Son simples de crear y entender.
2. Son poco costosos y no requieren mucho entrenamiento.

3. Son efectivos.

Actualmente las técnicas basadas en escenario cuentan con dos instrumentos de evaluación relevantes: Árbol de utilidad y Perfiles.

Árbol de Utilidad

Es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

La intención del uso del Árbol de Utilidad es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol.

Perfiles

Un perfil es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados.

Los *perfiles completos* definen todos los escenarios relevantes como parte del perfil. Esto permite al ingeniero de software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, puesto que incluye todos los posibles casos. Su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad.

Los *perfiles seleccionados* se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo con algunos requerimientos.

3.6.2 Evaluación basada en simulación.

Establece que la evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación del contexto del sistema donde se supone va a ejecutarse. La finalidad

es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad.

3.6.3 Evaluación basada experiencia.

Establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia trascendental; basada en factores subjetivos como la intuición y la experiencia. La mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación. Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

3.6.4 Evaluación basada en modelos matemáticos.

Establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro.

Técnica de evaluación propuesta.

De las técnicas antes mencionadas se utilizará la basada en escenario, pues los escenarios permiten de una manera factible concretar y entender los atributos de calidad. Igualmente la técnica es poco costosa, no requiere de mucho entendimiento pues es fácil de aplicar y efectiva. Se usará como instrumento de evaluación perfiles pues permite detallar con más precisión el requerimiento para un atributo de calidad.

3.7 Métodos de evaluación.

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. Los métodos de evaluación sirven para identificar los atributos de calidad con los que cumple el software, los cuales serán evaluados con las técnicas. Por esta razón, resulta factible estudiar alguno de los métodos de evaluación de AS definidos hasta el momento.

3.7.1 Software Architecture Analysis Method (SAAM).

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. (Erika Camacho, 2004)

La entrada principal de este método, es la de descripción de la arquitectura a ser evaluada, las salidas de la evaluación del método SAAM son las siguientes:

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.
- Entendimiento de la funcionalidad del sistema.

Como objetivo al aplicar este método es la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, partiendo de los requerimientos de modificabilidad.

En la siguiente tabla se muestran los pasos a seguir para evaluar una arquitectura utilizando SAAM.

1. Desarrollo de escenarios	Un escenario es una breve descripción de usos anticipados o deseados del sistema. De igual forma, estos pueden incluir cambios a los que puede estar expuesto el sistema en el futuro.
2. Descripción de la arquitectura	La arquitectura debe ser descrita haciendo uso de alguna notación arquitectónica que sea común a todas las partes involucradas en el análisis. Deben incluirse los componentes de datos y conexiones relevantes, así como la descripción del comportamiento general del sistema. El desarrollo de escenarios y la descripción de la arquitectura son usualmente llevados a cabo de forma intercalada, o a través de varias iteraciones.

3. Clasificación y asignación de prioridad de los escenarios	La clasificación de los escenarios puede hacerse en dos clases: directos e indirectos. Un escenario directo es el que puede satisfacerse sin la necesidad de modificaciones en la arquitectura. Un escenario indirecto es aquel que requiere modificaciones en la arquitectura para poder satisfacerse. Los escenarios indirectos son de especial interés para SAAM, pues son los que permiten medir el grado en el que una arquitectura puede ajustarse a los cambios de evolución que son importantes para los involucrados en el desarrollo.
4. Evaluación individual de los escenarios indirectos	Para cada escenario indirecto, se listan los cambios necesarios sobre la arquitectura, y se calcula su costo. Una modificación sobre la arquitectura significa que debe introducirse un nuevo componente o conector, o que alguno de los existentes requiere cambios en su especificación.
5. Evaluación de la interacción entre escenarios	Cuando dos o más escenarios indirectos proponen cambios sobre un mismo componente, se dice que interactúan sobre ese componente. Es necesario evaluar este hecho, puesto que la interacción de componentes semánticamente no relacionados revela que los componentes de la arquitectura efectúan funciones semánticamente distintas. De forma similar, puede verificarse si la arquitectura se encuentra documentada a un nivel correcto de descomposición estructural.
6. Creación de la evaluación global	Debe asignársele un peso a cada escenario, partiendo de su importancia relativa al éxito del sistema. Esta asignación de peso suele hacerse con base en las metas del negocio que cada escenario soporta. En el caso de la evaluación de múltiples arquitecturas, la asignación de pesos puede ser utilizada para la determinación de una escala general.

Tabla 2 Pasos contemplados por el método de evaluación SAAM. (Erika Camacho, 2004)

3.7.2 Architecture Trade-off Analysis Method (ATAM).

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos. Este método consta de 4 fases y 9 pasos que se muestran en la tabla siguiente.

Fase 1: Presentación	
1. Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
4. Identificación de los enfoques arquitectónicos	Estos elementos son detectados, pero no analizados.
5. Generación del Utility Tree	Se elicitan los atributos de calidad que engloban la “utilidad” del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de

	balance.
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 3 Pasos del método de evaluación ATAM. (Erika Camacho, 2004)

3.7.3 Active Reviews for Intermediate Designs (ARID).

El método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. ARID consta con 2 fases y 9 pasos para la evaluación de la arquitectura que se muestran en la siguiente tabla. (Erika Camacho, 2004)

Fase 1: Actividades Previas	
1. Identificación de los encargados de la revisión	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.
2. Preparar el	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos

informe De diseño	del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
3. Preparar los escenarios base	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
4. Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
Fase 2: Revisión	
5. Presentación del ARID	Se explica los pasos del ARID a los participantes.
6. Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.
7. Lluvia de ideas y establecimiento de prioridad de escenarios	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
8. Aplicación de los escenarios	Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos: Se agota el tiempo destinado a la revisión. Se han estudiado los escenarios de más alta prioridad. El grupo se siente satisfecho con la conclusión alcanzada. Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no

	conveniente, cuando el grupo encuentra problemas o deficiencias.
9. Resumen	El facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

Tabla 4 Pasos del método de evaluación ARID (Erika Camacho, 2004).

Método propuesto.

Un método de evaluación no es mejor que otro, sino que evalúa mejor en ciertas condiciones, un atributo de calidad dado. Por lo que se concluye que en dependencia de las condiciones y lo que se desea evaluar, será el método de evaluación empleado.

3.8 Evaluando la arquitectura propuesta.

A continuación se explican los atributos de calidad con los que cumple la arquitectura propuesta:

Eficiencia

No usa recursos de red ni necesita procesadores demasiado potentes ni con gran capacidad de almacenamiento como servidores. A la hora de diseñar esta arquitectura se tuvo en cuenta el consumo de memoria y el tiempo del procesador, cumple con este parámetro puesto que la aplicación realiza sus funciones de render en 9 milisegundos para una resolución de 640x480 a 30 FPS (Frame Por Segundos), dicha aplicación está por encima de la velocidad de captura de la cámara teniendo como promedio aproximadamente 85 FPS, esta prueba se realizó sin ningún marcador. Se recomienda hacer pruebas con uno o más marcadores.

Mantenibilidad

Este atributo constituye un buen medidor de la robustez y flexibilidad del producto, y son fundamentales las decisiones arquitectónicas que se tomen. La arquitectura que se analiza en este trabajo tiene como característica fundamental su extensibilidad. El hecho de que esté basada en componentes con interfaces bien definidas posibilita la sustitución de alguno de estos componentes con la menor influencia en los

demás. Si proporciona error permite hacerle modificaciones sin complicaciones posee un diseño claro debido al uso de patrones de diseño.

Portabilidad

Hasta ahora la aplicación que se ha realizado siguiendo la arquitectura propuesta no se ha compilado en GNU Linux, pero las herramientas de desarrollo son todas multiplataforma. Una de las decisiones importantes en aras de que los productos sean portables fue la selección de la tecnología de implementación. Para el desarrollo del sistema se utiliza el lenguaje de programación C++ con este lenguaje se puede desarrollar aplicaciones para cualquier sistema operativo. El hecho de la separación de la lógica de la aplicación de la interfaz de usuario garantiza que se pueda realizar cambios y estas modificaciones no afecten a los usuarios y se realicen de manera abstracta para los clientes.

Funcionalidad

La adecuación se define como la capacidad del producto de software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados. En la arquitectura propuesta se dieron solución a las propuestas hecha por los usuarios y se tuvieron en cuenta todos los requisitos funcionales y no funcionales que tendría el software. La arquitectura desarrollada presenta una buena adecuación puesto que en ella está soportado todo lo necesario para desarrollar una aplicación de Realidad Aumentada. La arquitectura propuesta da cumplimiento a los requisitos funcionales entre ellos están: la reproducción de video, dibujar la escena aumentada, seleccionar el recurso virtual en la escena aumentada, interactuar con la escena aumentada y cargar recurso en la escena, requisitos que son soportados en la arquitectura propuesta y da la medida de la adecuación del producto.

La interoperabilidad es la capacidad del producto software para interactuar con uno o más sistemas especificados, o sea, la manera de poder enviar, compartir y manipular información con otros sistemas, en este parámetro podemos resaltar elementos como la manera en que son manejados los ficheros que se guardan en la aplicación los cuales son en formato XML, que es un modo de lenguaje de texto compatible y manejado por la mayoría de las aplicaciones hoy en día,

Fiabilidad

Si ocurre alguna falla en la aplicación son tratados los errores, en caso de alguno se le muestra la información necesaria al usuario y la aplicación continua ejecutándose, en el caso de la gestión de

memoria está basada en un puntero (apuntador inteligente) que es un recolector de basura que implementaron los desarrolladores de OSG para gestionar la memoria utilizando referencias cruzadas, Otro aspecto importante es la madurez de las herramientas que se utilizan para implementar el sistema que soporta esta arquitectura, como por ejemplo QT donde se han hecho variedades de aplicaciones, es público y patrocinado por NOKIA, también la herramienta ARToolkit, que es abierta, versión comunitaria, estable con más de 10 años de desarrollo, mantenida por la comunidad y OSG que lleva más de 10 años de desarrollo, se utiliza en infinidad de proyectos desde juegos hasta la visualización científica.

Usabilidad

Se piensa que la usabilidad es un atributo al que se le da solución en la capa de presentación y no tiene más influencia en el resto de la arquitectura. Del mismo modo, son muchas las veces en que hay que decirle al cliente que no se puede hacer algo relacionado con la usabilidad porque afectaría cosas que van más allá de la capa de presentación. Si en algún momento tiene que ser sustituida la interfaz de usuario no se perdería el núcleo funcional de la aplicación de Realidad Aumentada. Otro aspecto importante está relacionado con darle información al usuario de la naturaleza de los errores que pueda presentar el programa, con este fin se definió una estrategia de tratamiento de errores, cuando ocurre un error el usuario es notificado. La facilidad con que se puede manejar la aplicación se debe en gran medida a la utilización de controles de usuario para representar los elementos que aparecen en la escena. Además se provee de una apariencia agradable al sistema.

3.9 Conclusiones parciales.

La arquitectura propuesta para el desarrollo de aplicaciones de RA es adecuada y logra el objetivo para el cual fue creada. Por tanto, se puede desarrollar nuevos productos de RA rigiéndose por esta primera versión de la arquitectura ya que la misma cumple con los atributos de calidad que expone la ISO 9126 para medir la calidad de software como se muestra en este capítulo.

CONCLUSIONES

Se hizo un estudio detallado de los principales aspectos de la descripción arquitectónica. Se caracterizó los principales estilos arquitectónicos, sus componentes, configuraciones y restricciones impuestas por los requisitos de los clientes. Dada las restricciones impuestas por algunos requisitos no funcionales del cliente y en función de los principales atributos de calidad de la Arquitectura de Software se seleccionaron las principales tecnologías y herramientas a utilizar en el desarrollo del sistema. La Arquitectura de Software propuesta cumple con: satisfacer los requisitos de los clientes, es “construible” por los desarrolladores, es “verificable” puede ser probada por los revisores y es “administrable” por los administradores del sistema. La arquitectura diseñada es reusable porque permite la creación de nuevos SRA con características semejantes, con tiempos de desarrollado relativamente cortos y sin cambios significativos a la arquitectura.

Todos los resultados obtenidos y analizados son positivos a pesar de que las métricas de arquitectura no son medibles numéricamente, evaluando así a la arquitectura como aceptable. Se diseñó una arquitectura base para SRA, sobre la cual se pueden ir agregando nuevos componentes y funcionalidades la cual cumple con los requisitos funcionales y no funcionales del sistema al cual da solución informática, por tanto, se cumplió con el objetivo de diseñar una arquitectura robusta, flexible y reusable.

RECOMENDACIONES

Como resultado del trabajo realizado se diseñó una arquitectura funcional, flexible y reusable, por estas características se recomienda que:

- Continuar trabajando en esta arquitectura en aras de perfeccionarla y agregarle nuevos componentes según sea necesario.
- Realizar pruebas de rendimiento con marcadores para optimizar el consumo de recursos.
- Migrar hacia Software Libre para que el sistema sea multiplataforma.

REFERENCIAS BIBLIOGRÁFICAS

1. **Ivar Jacobson, Grady Booch, James Rambaugh.** *El Proceso Unificado del Desarrollo de Software.* Madrid : Pearson Educacion S.A., 2000. 84-7829-036-2.

2. **Clements, Paul.** *A Survey of Architecture Description Languages. Proceedings of the International Workshop on Software Specification and Design.* Alemania : s.n., 1996.

3. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal.** *Pattern – Oriented Software Architecture. A System of Patterns.* Inglaterra : s.n., 1996.

4. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.** *Arquitectura de Software Guía de Estudio.* 2004.

5. **Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Roert Nord, Judith Stafford.** *Documenting Software Architecture Views and Beyond.* 2004. 0202703726.

6. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns, Elements of Reusable Object-Oriented Software.*

7. **David Garlan, Mary Shaw.** *An introduction to Software Architecture.* Pittsburgh : s.n., 1994.

8. **Rucker, Rudy.** *Software Engineering and Computer Games.* s.l. : Addison Wesley, 2002. 0201767910.

9. **Mary Shaw, David Garlan.** *Introduction to Software Architectures. New perspectives on an emerging discipline.* Prentice Hall : s.n.

10. UML tool, business process modeler and database designer for software development team. [Online] Visual Paradigm. [Cited: marzo 20, 2010.] http://www.visual-paradigm.com/aboutus/10reasons.jsp.

11. Qt - A cross-platform application and UI framework. [Online] [Cited: Marzo 25, 2010.] http://qt.nokia.com/products/.

12. Realidad Aumentada. [Online] [Cited: Marzo 20, 2010.] http://realidadeaumentada.com.br/home/index.php?option=com_content&task=view&id=6&Itemid= 28..

13. OpenSceneGraph. [Online] [Cited: Marzo 26, 2010.] http://www.openscenegraph.org/projects/osg/wiki/About/Introduction.

14. **Romero, Mario Orlando Pérez Fabienny y.** *Interacción entre elementos virtuales a través de estándares XML y X3D.* Ciudad de la Habana : s.n., 2008.

BIBLIOGRAFÍA

1. Lafuente, D.M. *Patrones de diseño*. 2006; Available from: www.minid.net/2006/01/02/patrones-de-diseno.

2. Clements, P., et al., *Documenting Software Architecture Views and Beyond*. 2004.

3. Goodliffe, P., *Code Craft: The Practice of Writing Excellent Code*. 2007: William Pollock.

4. Barbacci, M., et al. 1995; Available from: http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html.

5. Camacho, E., F. Cardeso, and g. Nuñez, *Arquitectura de Software Guía de Estudio*. 2004.

6. Crnkovic, I. and M. Larsson, *Building reliable component-based software systems*. 2002: Artech House.

7. Garrido, R. and A.G. Alonso, Técnicas de Interacción para Sistemas de Realidad Aumentada.

8. Bass, Len, Clements, Paul and Kazman, Rick. Software Architecture in Practice, Second Edition. s.l. : Addison Wesley, April 11, 2003. 0-321-15495-9.

9. Larman, Craig. UML y Patrones. 1999.

10. Amaya Flores, Elvira, Estrada Espinosa, Carlos y Manrique Ramirez, Joel. Resumen del libro "Lenguaje Unificado de Modelado" Disponible en: http://azul.bnct.ipn.mx/Entidades_vinculadas/Resumen_uml.doc. s.l. : ADISON WESLEY, 2003.

11. Las 4+1 Vistas.http://synergix.wordpress.com/2008/07/31/las-4-mas-1-vistas/.

12 Buschmann, Frank. "Pattern-Oriented Software Architecture, Volume 1: A System of Patterns". 1996.

13. Oktaba, Hanna. "Introducción a Patrones". Mexico : s.n.

14. Clements Paul, Kazman Rick, Klein Mark. “Evaluating Software Architectures: Methods and Case Studies”. ISBN 0-201-70482-X.

ANEXOS

Anexo 1.

Caso de Uso:	Crear Asociaciones de Recurso
Actores:	Usuario
Resumen:	Este caso de uso comienza cuando el actor desea asociar algún recurso y un dato de registro.
Referencias	RF4
Prioridad	Crítico
Flujo Normal de Eventos	
Sección "Principal"	
Acción del Actor	**Respuesta del Sistema**
1- Selecciona el menú Crear Asociaciones. 2- Selecciona el recurso y dato de registro que desea asociar.	1- El sistema muestra los recursos y los datos de registros existentes. 2- Muestra el recurso asociado.

Tabla 5 Descripción textual del Caso de Uso Crear Asociaciones de Recursos.

<table>
<tr><td>Caso de Uso:</td><td colspan="2">Cargar Recurso en Escena</td></tr>
<tr><td>Actores:</td><td colspan="2">Usuario</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de Uso se inicia cuando el usuario decide cargar un recurso para representar en la escena y culmina cuando se muestra el recurso.</td></tr>
<tr><td>Precondiciones:</td><td colspan="2">Se conoce la dirección del fichero de configuración.

El fichero de configuración está editado.

Está creado manejador de elementos virtuales y el manejador de patrones de los marcadores.</td></tr>
<tr><td>Referencias</td><td colspan="2">RF 2, RF2.1, RF2.2</td></tr>
<tr><td>Prioridad</td><td colspan="2">Crítico</td></tr>
<tr><td colspan="3">Flujo Normal de Eventos</td></tr>
<tr><td colspan="3">Sección “Principal”</td></tr>
<tr><td colspan="2">Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2">1- El usuario selecciona en la interfaz de usuario el menú para cargar recursos.

2- Selecciona el recurso deseado.</td><td>1- Muestra los posibles recursos a cargar:

a) Modelos 3D.

b) Archivos.

2- Carga el recurso seleccionado.</td></tr>
<tr><td>Poscondiciones</td><td colspan="2">Se muestra el recurso</td></tr>
</table>

Tabla 6 Descripción textual del Caso de Uso Cargar Recurso en la Escena.

Caso de Uso:	Interactuar con Escena Aumentada	
Actores:	Usuario	
Resumen:	El CU es iniciado por el usuario. Al oprimir una tecla, de acuerdo con la tecla que oprime, se determinara en qué dirección y sentido se desplazará el objeto virtual con respecto al marcador.	
Precondiciones:	La escena aumentada tiene que haberse mostrado.	
Referencias	RF 6	
Prioridad	Crítico	
Flujo Normal de Eventos		
Sección "Principal"		
Acción del Actor		**Respuesta del Sistema**
1- Oprime una tecla		1.1- Capturar tecla oprimida. 1.2- Comprueba que sea una de las teclas especificadas para desplazar el objeto virtual. 1.3- Desplaza el objeto en la dirección y sentido seleccionado.
Poscondiciones	El objeto es trasladado.	

Tabla 7 Descripción textual del Caso de Uso Interactuar con Escena Aumentada.

Caso de Uso:	Reproducir Video
Actores:	Usuario
Resumen:	Este caso de uso comienza cuando el actor desea reproducir un video de la escena aumentada y culmina cuando obtiene la visualización del video.
Precondiciones:	Conocer la dirección del archivo. Cámara conectada a la PC. Conocer URL del archivo de video.
Referencias	RF 1, RF1.1, RF1.3, RF 1.4
Prioridad	Crítico

Flujo Normal de Eventos	
Sección "Principal"	
Acción del Actor	**Respuesta del Sistema**
1- Seleccionar en la interfaz de usuario, el menú de Reproducir Video. 2- Selecciona la opción deseada (Ver sección según corresponda).	1-Muestra opción de Reproducir Video: a) Reproducir Video desde Archivo. b) Reproducir Video desde Remoto. c) Reproducir Video desde Cámara.
Sección "Reproducir Video desde Archivo"	

Acción del Actor	**Respuesta del Sistema**
1- Busca el video deseado.	1- El sistema muestra un cuadro de diálogo donde el usuario debe seleccionar la dirección del archivo y el formato de video. 2- Comienza la reproducción.
Sección "Reproducir Video desde Remoto"	
Acción del Actor	**Respuesta del Sistema**
1- Busca el video deseado.	1- El sistema muestra un cuadro de diálogo donde el usuario debe seleccionar la URL o dirección remota donde se encuentra la fuente de video (servidor de streaming). 2- Comienza la reproducción.
Sección "Reproducir Video desde Cámara"	
Acción del Actor	**Respuesta del Sistema**
	1- El sistema muestra un cuadro de diálogo donde el usuario debe

1- Ajusta los parámetros de la cámara	seleccionar la resolución de la cámara, velocidad de captura y formato del archivo de video. 2- Comienza la reproducción.
Poscondiciones	Se visualiza el archivo de video solicitado.

Tabla 8 Descripción textual del Caso de Uso Reproducir Video.

GLOSARIO DE TÉRMINOS

Árbol de Utilidad: es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

Captura: Proceso por el cual se obtiene información de un dispositivo externo (Cámara, webcam, etc.) y se almacena o se observa en la computadora.

CASE: (Computer Aided Software Engineering): se incluyen una serie de herramientas, lenguajes y técnicas de programación que permiten la generación de aplicaciones de manera semiautomática.

GRASP (General Responsability Assignment Software Patterns): son patrones generales de software para asignación de responsabilidades.

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

Lenguaje de Modelado Unificado (UML): es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software

Metodología: En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo.

POO (Programación Orientada a Objetos): es un paradigma de programación, que permite descomponer un problema en subgrupos relacionados. Cada subgrupo pasa a ser un objeto auto contenido que contiene sus propias instrucciones y datos que le relacionan con ese objeto.

Proceso de desarrollo de software: es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.

Rational Unified Process (RUP): es un proceso de desarrollo de software.

Streaming: Tecnología que permite la reproducción de sonido o vídeo sin que sea necesario descargar previamente todo el archivo de recurso.

XML: siglas en inglés de Extensible Markup Language (Lenguaje de Etiquetado Extensible). Es un lenguaje basado en SGML, capaz de describir cualquier tipo de información en forma personalizada.