Facultad 8

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

TÍTULO: Búsqueda de los elementos arquitectónicos del software educativo en la UCI.

AUTOR: Leonard Fundora Echevarria.

TUTOR: Ing. Dianelys Espinosa Zaldívar.

CO-TUTOR: M. Sc. Febe Ángel Ciudad Ricardo.

ASESOR: Lic. Lesyanis Pompa Arcia.

Julio, 2008

"Año 50 de la Revolución"

“El mundo camina hacia la era electrónica...Todo indica que esta ciencia se constituirá en algo así como una medida del desarrollo; quien la domine será un país de vanguardia. Vamos a volcar nuestros esfuerzos en este sentido con audacia revolucionaria”

Ernesto Che Guevara.

DECLARACIÓN DE AUTORÍA

Declaramos ser los únicos autores de la presente tesis y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmó la presente a los ____ días del mes de _____ del año _____.

Leonard Fundora Echevarria	Ing. Dianelys Espinosa Zaldívar
_____	_____
Firma del Autor	Firma del Tutor

DEDICATORIA

Dedico este trabajo especialmente:

A mi madre, que es la razón por la cual me esfuerzo cada día a ser mejor, por darme su apoyo incondicional en tantos años de estudio y momentos difíciles a los que me he enfrentado; a mi padre, a mis hermanos, tías y tíos, que mucho me han ayudado para llegar a ser lo que soy hoy en día; muy en especial a mi abuela que ya no está entre nosotros, pero mucho de lo que soy y de lo que he conseguido se lo debo a ella y a mis amigos que con su apoyo me alentaron en el desarrollo de la Tesis.

AGRADECIMIENTOS

A la Universidad de las Ciencias Informáticas, por haberme formado como profesional a la altura de nuestros tiempos.

A Dariel, Lesyanis, Mercedes y Aldin por ser tan buenos amigos y por sobrellevarme todos estos años, además de brindarme su ayuda cada vez que la he necesitado.

A mi tutora Ing. Dianelys Espinosa Zaldívar, que a pesar de que no poseía mucho conocimiento en el tema, supo orientarme y darme confianza.

A mis amigos de la infancia que a pesar de la distancia me tienen siempre presente y se han preocupado mucho por mí. En especial a Dany, Yaniel y Reina.

A mis compañeros de cinco años de estudio y esfuerzo, por brindarme su amistad desinteresada, y compartir tantas cosas buenas y malas, que durarán en mi memoria para siempre. En especial la gente del edificio 89.

A mis padres y familiares, por guiarme y apoyarme durante tantos años; y confiar en que podía lograrlo.

Leonard

RESUMEN

La Universidad de las Ciencias Informáticas (UCI) es una institución que se dedica al desarrollo de software, dentro de ellos, el Software Educativo (SWE) constituye una de sus principales líneas de producción. Actualmente esta producción se ve afectada por numerosas deficiencias que presenta el producto final; las cuales son provocadas, entre otras, por la no definición de los elementos arquitectónicos comunes y significativos para el desarrollo de los mismos. Es por ello que surge la necesidad de identificar una arquitectura que permita la representación de los elementos arquitectónicos del SWE en la UCI con el objetivo de encontrar y corregir los errores a tiempo. Para su creación se expuso de forma clara toda una gama de información acerca de los estilos y patrones arquitectónicos más conocidos, que permitió después de un análisis profundo la selección de un conjunto de aspectos que se utilizaron como guía para su elaboración, efectuándose luego la validación de la misma.

PALABRAS CLAVE

Software Educativo, Arquitectura de Software, Elementos Arquitectónicos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCION

Uno de los aspectos más importante del mundo contemporáneo es el acelerado desarrollo de la ciencia y la tecnología y dentro de está los avances alcanzados en la esfera de la producción de software educativo. En los últimos años los modelos de arquitecturas utilizadas en el diseño del software educativo han sido en forma desorganizada y poco documentada. En tal sentido el aumento exponencial que sufrirá este fenómeno en los próximos años, crea la necesidad de perfeccionar las notaciones y lenguajes existentes que contribuyan a mejorar el desarrollo de este tipo de aplicaciones, mediante los métodos, procedimientos y herramientas que provee la Ingeniería de Software para construir programas educativos de calidad, siguiendo las pautas de las teorías del aprendizaje y de la comunicación subyacentes.

Como elemento de vital importancia en el diseño de software educativo se encuentra la utilización de una adecuada y conveniente arquitectura de software (en lo adelante AS), Shaw y Garlan **(1)**, en su libro de referencia sobre la materia, tratan la AS de la siguiente forma: *Incluso desde que el primer programa fue dividido en módulos, los sistemas de software han tenido arquitecturas, y los programadores han sido responsables de sus interacciones a través de módulos y de las propiedades globales de ensamblaje. Históricamente, las arquitecturas han estado implícitas -bien como accidentes en la implementación, bien como sistemas legados del pasado-. Los buenos desarrolladores de software han adoptado, a menudo, uno o varios patrones arquitectónicos como estrategias de organización del sistema, pero utilizaban estos patrones de modo informal y no tenían ningún interés en hacerlos explícitos en el sistema resultante.*

Cada vez que se narra la historia de la AS, se reconoce que en un principio, hacia 1968, Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera. **(2)**

Aunque Dijkstra no utiliza el término "arquitectura" para describir el diseño conceptual del software, sus conceptos sientan las bases para posteriores definiciones como la reconocida de Clements **(3)**: 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 la 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.

Considerada como disciplina por mérito propio, la AS ha de ser beneficiosa como marco de referencia para satisfacer requerimientos, una base esencial para la estimación de costos y administración del proceso y para el análisis de las dependencias y la consistencia del sistema.

En la actualidad, la producción de software educativo tanto en el mundo como en Cuba, ha tenido gran aceptación e implementación; tal es el caso de centros laborales y educacionales como: Empresa de Tecnologías de la Información y Servicios Telemáticos Avanzados (SITMATEL), SIS-COPEXTEL, Instituto Superior Politécnico José Antonio Echeverría (CUJAE). Esto ha motivado el interés por investigar cual ha sido el comportamiento de la UCI, de la implementación de software educativos y en especial de los elementos estructurales y lógicos de las arquitecturas de desarrollo de estas aplicaciones.

Las condiciones en las cuales se desenvuelve la producción de software educativo a nivel internacional y de forma particular en la universidad, conlleva a que se haya identificado como **problema científico**: *Insuficiente conocimiento de los elementos estructurales y lógicos comunes; así como significativos del software educativo en la UCI.*

Para resolver el problema referenciado anteriormente se hizo necesario que el **objetivo general** de esta investigación esté dirigido a *elaborar una propuesta de los elementos estructurales y lógicos (arquitectónicos) a utilizar en la producción de software educativo en la UCI*.

Y como **objetivos específicos:**

1. *Definir los componentes conceptuales comunes, significativos y con capacidad de generalización de los elementos estructurales y lógicos del software educativo producido en la UCI.*
2. *Definir los elementos arquitectónicos necesarios en la producción del software educativo en la UCI.*

Para darle cumplimiento a dicho objetivo fue necesario centrar la investigación en el *análisis y diseño del software educativo en la UCI*, lo cual constituye el **objeto de estudio.**

El **campo de acción** será el *proceso de concepción y diseño pedagógico del software educativo en la UCI*.

Para la realización de la investigación; la misma fue sustentada en la siguiente **hipótesis**; *la aplicación de los elementos estructurales y funcionales (arquitectónicos) significativos en la producción del*

software educativo en la UCI, permitirá un mejor diseño funcional y pedagógico así como una disminución del tiempo de desarrollo de este tipo de aplicación.

Para darle cumplimiento al objetivo trazado se determinó que las **tareas** a realizar en esta investigación estarían dirigidas a:

1. *Estudiar el estado actual de la producción de software educativo en Cuba.*
2. *Estudiar la documentación generada a partir del diseño y concepción pedagógica del software educativo en la UCI.*
3. *Estudiar los elementos estructurales, funcionales y de arquitectura significativos en la producción de software educativo en el mundo actualmente.*
4. *Identificar los elementos estructurales, funcionales y de arquitectura significativos como resultado de los diferentes procesos de diseño y concepción pedagógica utilizados en la UCI.*
5. *Identificar los elementos estructurales, funcionales y de arquitectura significativos como resultado de los diferentes procesos de diseño y concepción pedagógica utilizados en el mundo.*
6. *Proponer los elementos estructurales y lógicos a utilizar en la producción de software educativo en la UCI.*
7. *Validar según el criterio experto la aplicación de los elementos propuestos.*

Esta propuesta trae como objetivo un mejoramiento en el costo de tiempo del producto, además permitirá ir perfeccionando el proceso de desarrollo del SWE y la organización interna de cada uno de los proyectos que sigan esta línea de producción en la Universidad.

Algunos de los métodos que se utilizaron en la investigación son:

- **Analítico – Sintético**, para el "análisis" de los elementos de la situación problemática con el propósito de descomponer dicho problema en elementos por separado para un mejor entendimiento del software educativo y la "síntesis" para describir y resumir el proceso que se desarrollará a partir de la investigación en la UCI.
- **Análisis Histórico – Lógico**, para investigar la existencia de trabajos anteriores referentes al software educativo, para usarlos como puntos de referencia en la investigación y compararlos con los resultados alcanzados.

- **Entrevistas y Encuestas** a líderes y estudiantes de proyecto de software educativo en la UCI, así como a directivos de producción educativa en la Universidad, para recopilar toda la información que estas nos puedan suministrar, para conocer cómo se desarrolla el proceso de desarrollo de software educativo de forma general en la universidad.

El presente documento se conforma por 3 capítulos, a continuación se presenta el nombre del capítulo y su objetivo en un contexto global:

Capítulo 1: Fundamentación Teórica. En este capítulo se formalizan todos los conceptos asociados al tema y que son necesarios para la comprensión de lo que se describe en el resto del trabajo. Además se analizaron algunas soluciones que sirvieron de base para la solución propuesta.

Capítulo 2: Descripción de la propuesta de solución. En este capítulo se presenta la propuesta de arquitectura para el proceso de desarrollo del SWE en la UCI, en la cual se describen cada uno de los elementos que componen el modelo, especificándose también las ventajas que trae su aplicación.

Capítulo 3: Validación de la propuesta de solución. En este capítulo se determina la probabilidad de éxito que tenga la propuesta, para esto se realizan un conjunto de cálculos los cuales se detallan con el objetivo de lograr un mayor entendimiento.

Se considera que el tema objeto de estudio es una modesta contribución al esfuerzo que se realiza por convertir a la UCI en uno de los centros de referencia del desarrollo tecnológico a nivel internacional, así como obtener una arquitectura para la producción de los SWE, modular, flexible, que satisfaga con los requisitos (Analistas) y que pueda ser construible (Diseñadores y Programadores). Que cumpla con los parámetros de funcionalidad, eficiencia y confiabilidad.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

La necesidad de una adecuada arquitectura es una preocupación cada vez mayor en el ámbito informático, cuyos resultados inmediatos se aprecian en la realización de un software y posteriormente en la aceptación de éste por los usuarios. El objetivo principal de la Arquitectura de Software es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto.

En el presente capítulo se abordarán los elementos que servirán de fundamento teórico a la investigación presentada. Conceptos y teorías sobre, Software Educativo Arquitectura de Software, Estilo Arquitectónico, Patrón Arquitectónico serán trabajados con detenimiento para servir como fundamentación científica de lo investigado.

1.2 Software Educativo

"Las tecnologías de la información y la comunicación (TIC) ofrecen grandes posibilidades al mundo de la educación. Pueden facilitar el aprendizaje de conceptos y materias, ayudar a resolver problemas y contribuir a desarrollar las habilidades cognitivas. Las áreas de aplicación de todas estas técnicas, englobadas en lo que normalmente se denominan informática educativa, son tanto la enseñanza reglada, comúnmente denominada curricular, como la formación en todos los ámbitos posibles. De esta manera, se nos presenta la posibilidad de aprovechar la tecnología para crear situaciones de aprendizaje y enseñanza novedosas. "**(4)**

La definición de software educativo es expresada como: "Programas de ordenador creados con la finalidad específica de ser utilizados como medio didáctico, es decir, para facilitar los procesos de enseñanza y de aprendizaje". **(5)**

1.2.1 Características de los Software Educativos

Los software multimedia educativos, permiten agrupar una serie de factores presentes en otros medios, pero a la vez agregar otros hasta ahora inalcanzables:**(6)**

- Permite la interactividad con los estudiantes, retroalimentándolos y evaluando lo aprendido
- Facilita las representaciones animadas.

- Incide en el desarrollo de las habilidades a través de la ejercitación. Permite simular procesos complejos.
- Reduce el tiempo que se dispone para impartir gran cantidad de conocimientos facilitando un trabajo diferenciado, introduciendo al estudiante en el trabajo con los medios computarizados.
- Facilita el trabajo independiente y a la vez un tratamiento individual de las diferencias.
- Permite al usuario (estudiante) introducirse en las técnicas más avanzadas.
- Posibilidades de estudiar procesos que no es posible observar directamente.
- Autocontrol del ritmo de aprendizaje.

Además presentan otras características que facilitan el logro de sus objetivos que atienden a diversos aspectos funcionales, técnicos y pedagógicos. A continuación se mencionan algunos: **(7)**

- Facilidad de uso e instalación.
- Versatilidad (adaptación a diversos contextos).
- Calidad del entorno audiovisual.
- La calidad en los contenidos (bases de datos).
- Navegación e interacción.
- Originalidad y uso de tecnología avanzada.
- Capacidad de motivación.
- La documentación.

1.3 Arquitectura de Software

La arquitectura de software (AS) es el arte de proyectar y construir sistemas lógicos para computadoras; y como todo arte debe cultivarse y se adquiere con la experiencia. Establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Una AS se selecciona y diseña en base a objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, auditabilidad, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías, mientras que otras tecnologías no son aptas para determinadas arquitecturas. Por ejemplo, no es viable emplear una AS de tres capas para implementar sistemas en tiempo real.

La AS define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser impleméntable en una *arquitectura física*, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea.

Dicha arquitectura dirige el desarrollo de los sistemas software y contribuye a que estos se lleven a cabo en los límites establecidos de costos y tiempo, y fundamentalmente debe garantizar que se cumpla con los requisitos funcionales y no funcionales de los usuarios. La misma es el resultado del trabajo durante todo el ciclo de vida del proyecto, y principalmente en las primeras iteraciones, de un grupo de trabajo encabezado por el arquitecto (o grupo de arquitectura, en dependencia de las dimensiones del proyecto).

A continuación veremos las principales corrientes arquitectónicas:

- ***Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.*** Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código.
- ***Arquitectura*** como una ***Etapa de ingeniería y diseño orientada a objetos.*** Esta corriente es una alternativa de la descrita anteriormente. Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de UML y Rational. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan que la AS, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descritas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten. **(8) (9)**

- ***Arquitectura basada en patrones,*** esta corriente se basa principalmente en la redefinición de los estilos como patrones POSA, el diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura. **(10)(11)(12)**
- ***Arquitectura procesual y metodologías.*** Esta surge desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Félix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci y Charles Weinstock, Intenta establecer modelos de ciclo de vida y técnicas de diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la AS. **(13)**

1.3.1 Definición de Arquitectura de Software

La definición oficial de AS es la IEEE Std 1471-2000 que reza así: “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”.

Existen otras definiciones clásicas de la arquitectura tales como: "la estructura global del software y a las formas en que la estructura proporciona la integridad conceptual de un sistema". En su forma más simple, la arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera en que los componentes interactúan y la estructura de datos que van a utilizar los componentes. Sin embargo, en un sentido más amplio, los "componentes" se pueden generalizar para presentar los elementos principales del sistema y sus interacciones. **(14)**

1.3.2 Importancia de la Arquitectura de Software

En su libro dedicado a la AS, Bass y sus colegas **(15)** identifican tres razones claves por las que la AS es importante:

- las representaciones de la arquitectura de software facilitan la comunicación entre todas las partes (partícipes) interesadas en el desarrollo de un sistema basado en computadora.
- la arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería del software que sigue, y es tan importante en el éxito final del sistema como una entidad operacional.
- la arquitectura <<constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes>>.

1.4 Estilos Arquitectónicos

Estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. Estos "definen a una familia de sistemas en términos de un patrón de organización estructural. Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que puede ser usado así como un conjunto de restricciones de cómo pueden ser combinados". **(14)**

Los estilos de arquitectura se definen como las 4 C **(16)**:

- Componentes (Elementos)
- Conectores
- Configuraciones
- Restricciones (Constraints)

1.4.1 Estilos centrados en datos

En el centro de esta arquitectura se encuentra un almacén de datos (por ejemplo, un documento o una base de datos) al que otros componentes acceden con frecuencia para actualizar, añadir, borrar o bien modificar los datos del almacén. La figura 1 representa un estilo típico basada en los datos. El software de cliente accede a un almacén central. En algunos casos, el almacén de datos es pasivo. Esto significa que el software de cliente accede a los datos independientemente de cualquier cambio en los datos o de las acciones de otro software de cliente. Una variación en este acceso transforma el almacén en una <<pizarra>> que envía notificaciones al software de cliente cuando los datos de interés del cliente cambian. Las arquitecturas basadas en los datos promueven la capacidad de integración (integrability) **(15)**. Por consiguiente, los componentes existentes pueden cambiarse o los componentes del nuevo cliente pueden añadirse a la arquitectura sin involucrar a otros clientes (porque los componentes del cliente operan independientemente). Además, los datos pueden ser transferidos entre los clientes utilizando un mecanismo de pizarra por ejemplo, el componente pizarra sirve para transferencia de información entre clientes). Los componentes cliente son procesos ejecutados independientemente.

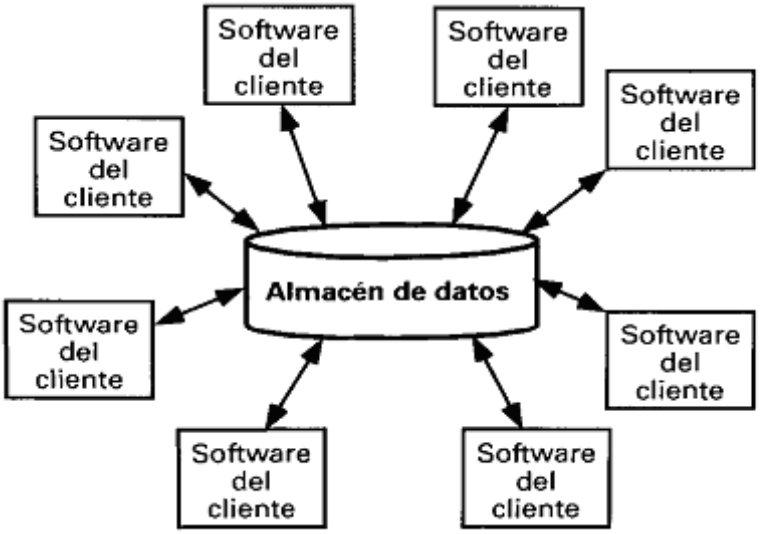


Figura 1: Arquitectura basada en los datos.

Dentro de este estilo arquitectónico se encuentra la arquitectura en pizarra:

- **Arquitectura en Pizarra**: La AS en *pizarra* es un modelo arquitectónico de software habitualmente utilizado en sistemas expertos, sistemas multiagente y, en general, sistemas basados en el conocimiento. La arquitectura en pizarra consta de múltiples elementos funcionales, denominados agentes, y un instrumento de control denominado pizarra. Los agentes suelen ser programas especializados en una tarea concreta o elemental. Todos ellos cooperan para alcanzar una meta común, si bien, sus objetivos individuales no están aparentemente coordinados. El comportamiento básico de cualquier agente consiste en examinar la pizarra, realizar su tarea y escribir sus conclusiones en la misma pizarra. De esta manera, otro agente puede trabajar sobre los resultados generados por el primero. La computación termina cuando se alcanza alguna condición deseada entre los resultados escritos en la pizarra. La pizarra tiene un doble papel. Por una parte, coordina a los distintos agentes y, por otra, facilita su intercomunicación. El estado inicial de la pizarra es una descripción del problema a resolver. El estado final será la solución del problema. Los resultados generados por los agentes deben responder a un lenguaje y semántica común. En general, se suelen utilizar formalismos lógicos o matemáticos, tales como expresiones lógicas de primer orden.

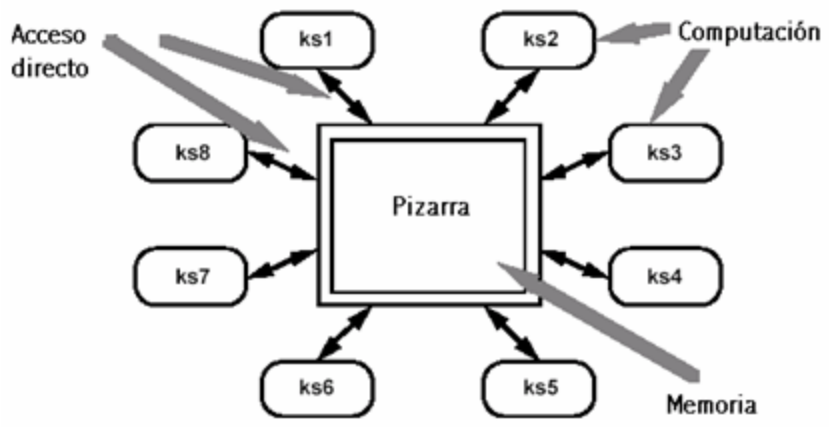


Figura 2: Pizarra [basado en (17)].

Ventajas e Inconvenientes de la Arquitectura en Pizarra

Esta arquitectura es tremendamente útil cuando el problema a resolver (o algoritmo a implementar) es *extremadamente complejo* en términos cognitivos. Es decir, cuando el flujo de control del algoritmo es enrevesado, o simplemente, no se tiene un conocimiento completo del problema a resolver.

Las desventajas de la arquitectura son bastante obvias a priori. Es importante no generalizar en este aspecto, puesto que cada implementación en particular puede solventar estas desventajas en algún ámbito limitado:

- No existe garantía de que se alcanzará una solución.
- Es una arquitectura ineficiente, puesto que no existe una cota respecto al tiempo de cómputo necesario para resolver el problema.
- Es difícil obtener una traza de los pasos que llevaron a la solución, es decir, no ofrece explicaciones.

Desde un punto de vista más filosófico, la arquitectura en pizarra ofrece un interesante experimento de tipo social. Cada agente tiene sus propios objetivos, desconoce los objetivos de los demás, y tampoco conoce el objetivo global (la solución del problema). Sin embargo, se produce una cooperación inconsciente entre ellos que lleva a una meta más importante.

1.4.2 Estilos de flujo de datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.

Esta arquitectura se aplica cuando los datos de entrada son transformados a través de una serie de componentes computacionales o manipulativos en los datos de salida. Un patrón tubería y filtro (Figura 3a) tiene un grupo de componentes, llamados filtros, conectados por tuberías que trasmiten datos de un componente al siguiente. Cada filtro trabaja independientemente de aquellos componentes que se encuentran en el flujo de entrada o de salida; está diseñado para recibir la entrada de datos de una cierta forma y producir una salida de datos (hacia el siguiente filtro) de una forma específica. Sin embargo, el filtro no necesita conocer el trabajo de los filtros vecinos.

Si el flujo de datos degenera en una simple línea de transformadores (Figura 3b) se le denomina secuencial por lotes. Este patrón aplica una serie de componentes secuenciales (filtros) para transformarlos.

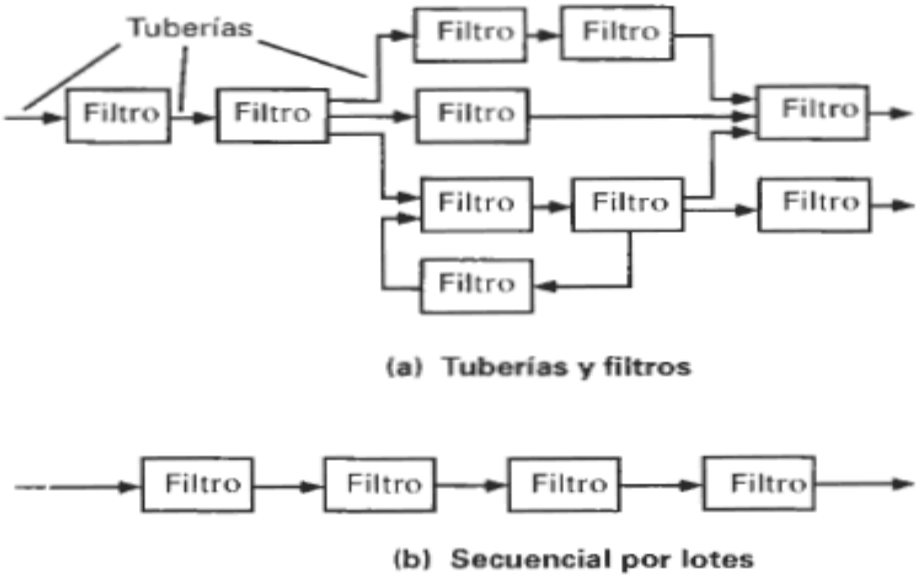


Figura 3: Arquitectura de flujo de datos.

1.4.3 Estilos de llamada y retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Existen dos subestilos dentro de esta categoría:

- arquitectura de programa principal/subprograma: esta estructura clásica de programación descompone las funciones en una jerarquía de control donde un programa <<principal>> llama a un número de componentes del programa, los cuales, en respuesta, pueden también llamar a otros componentes. La Figura 4 representa una arquitectura de este tipo.
- arquitectura de llamada de procedimiento remoto: los componentes de una arquitectura de programa principal/subprograma, están distribuidos entre varias computadoras de una red.

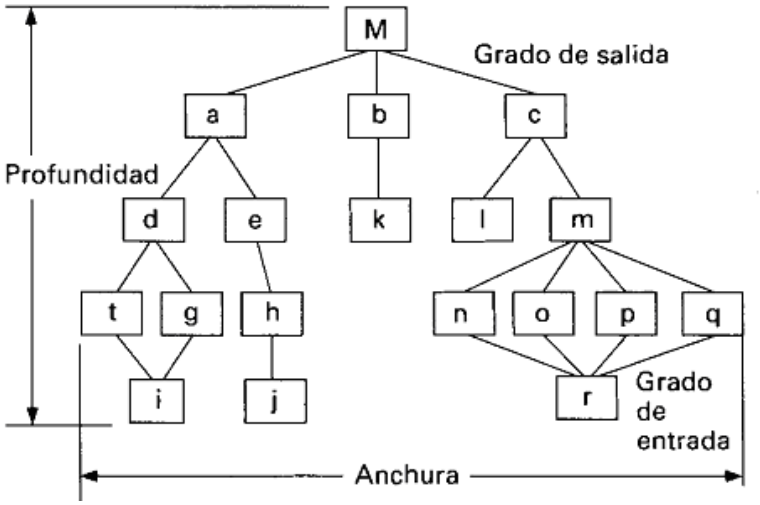


Figura 4: Terminologías de estructura para un estilo arquitectónico de llamada y retorno.

Dentro de este estilo arquitectónico se encuentran las siguientes arquitecturas:

- **Arquitectura Orientada a Objetos:** Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. En la caracterización clásica de David Garlan y Mary Shaw **(17)**, los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos.

- **Arquitectura basada en Componentes:** Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. Los componentes son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones, y conjuntamente con sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.
- **Arquitectura en Capas:** La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.

Capas o niveles

1.- Capa de presentación: es la que ve el usuario (hay quien la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser amigable (entendible y fácil de usar) para el usuario.

2.- Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

3.- Capa de datos: es donde residen los datos y es la encargada de acceder a los datos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

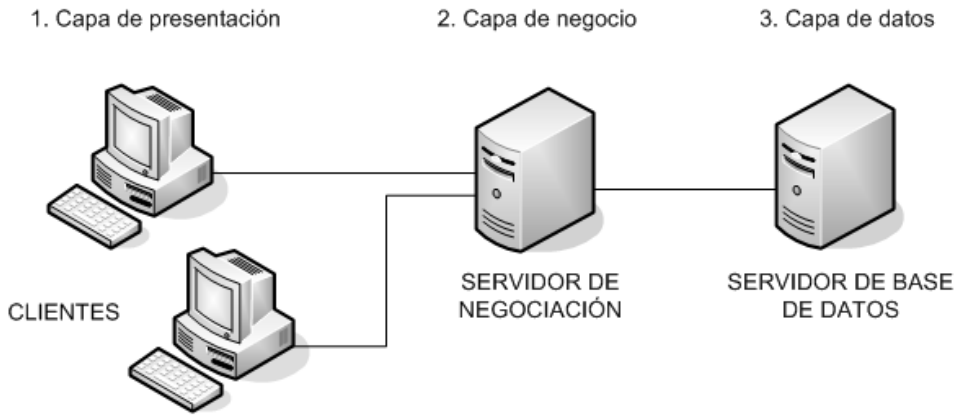


Figura 5: Arquitecturas de capas.

Ventajas de la Arquitectura de tres niveles o capas

La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio solo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería: Modelo de interconexión de sistemas abiertos.

En el diseño de sistemas informáticos actual se suele usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

1.4.4 Estilo de Código Móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando.

- **Arquitectura de Máquinas Virtuales:** Esta arquitectura se conoce como intérpretes basados en tablas ó sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen un extenso

espectro que está comprendido desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación.

1.4.5 Estilos Peer to Peer

Esta familia se conoce también como componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes.

- **Arquitecturas Basadas en Eventos:** Estas se han llamado también arquitectura de invocación implícita, estas se vinculan con sistemas basados publicación-suscripción. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciar mediante difusión uno o más eventos.
- **Arquitecturas Basadas en Recursos:** Esta define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URIs identifican los recursos y HTTP es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación.
- **Arquitecturas Orientadas a Servicios**: La Arquitectura Orientada a Servicios (en inglés Service-Oriented Architecture o SOA), es un concepto de AS que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.
  "Una arquitectura de aplicación en la cual todas las funciones se definen como servicios independientes con interfaces invocables bien definidas, que pueden ser llamadas en secuencias definidas para formar procesos de negocios" (IBM).

  SOA es una arquitectura de software que permite la creación y/o cambios de los procesos de negocio de forma ágil, a través de la composición de nuevos procesos utilizando las funcionalidades de negocio que están contenidas en la infraestructura de aplicaciones actuales o futuras (expuestas bajo la forma de webservices).

  SOA define las siguientes capas de software:

  - aplicativa básica, sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad.

- de exposición de funcionalidades, donde las funcionalidades de la capa aplicativas son expuestas en forma de servicios (webservices).
- de integración de servicios, facilitan el intercambio de datos entre elementos de la capa aplicativa orientada a procesos empresariales internos o en colaboración.
- de composición de procesos, que define el proceso en términos del negocio y sus necesidades, y que varia en función del negocio.
- de entrega, donde los servicios son desplegados a los usuarios finales.

Al contrario de las arquitecturas orientado a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación. La definición de la interfaz encapsula (oculta) las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo (como Plataforma Java o Microsoft.NET). Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio C Sharp podría ser usado por una aplicación Java.

Ventajas de la Arquitectura Orientada a Servicios

- Se obtendrán sistemas más flexibles y adaptables.
- Facilidades de mantenimiento, desde que las aplicaciones ya no son mas monolíticas, si no que compuestas de servicios.
- Se tendrá la habilidad de cambiar proveedores de servicios sin que se afecte la arquitectura, al mantener una interfaz estable.
- El uso de estándares en SOA permite la interoperabilidad entre plataformas y lenguajes de programación.
- Obtención de software de mejor calidad, usando componentes que han sido usados y probados por muchos usuarios, o usando nuestros propios servicios con un alto nivel de rehúso dentro de la empresa.
- La respuesta a las demandas de los clientes, y la competitividad en el mercado y contra la competencia se agilizará, al poder responder con tiempo y con la calidad requerida.

Los estilos arquitectónicos citados anteriormente son solo una pequeña parte de los que dispone el diseñador de software. Una vez que la ingeniería de requisitos define las características y las restricciones del sistema que ha de ser construido, se escoge el patrón arquitectónico (estilo) o la combinación de patrones (estilos) que mejor encajan con las características y restricciones. En muchos casos, puede ser apropiado más de un patrón y se podrían diseñar y evaluar estilos arquitectónicos alternativos

1.5 Modelos o Vistas de Arquitectura

Toda AS debe describir diversos aspectos del software. Generalmente, cada uno de estos aspectos se describe de una manera más comprensible si se utilizan distintos modelos o vistas. Es importante destacar que cada uno de ellos constituye una descripción parcial de una misma arquitectura y es deseable que exista cierto solapamiento entre ellos. Esto es así porque todas las vistas deben ser coherentes entre sí, evidente dado que describen la misma cosa.

Una vista arquitectónica es "una descripción simplificada (una abstracción) de un sistema desde una perspectiva particular o punto de vista, que cubre particularidades y omite entidades que no son relevantes a esta perspectiva". **(18)**

Ante el número y variedad de definiciones existentes de AS, se presentan los modelos existentes según Mary Shaw y David Garlan: **(17)**

1) Modelos estructurales: Sostienen que la AS está compuesta por componentes, conexiones entre ellos y otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizado por el desarrollo de lenguajes de descripción arquitectónica (ADLs).
2) Modelos de framework: Son similares a la vista estructural, pero su énfasis primario radica en la (usualmente una sola) estructura coherente del sistema completo, en vez de concentrarse en su composición. Los modelos de framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software específicas de dominios, como CORBA, o modelos basados en CORBA, o repositorios de componentes específicos, como PRISM.

3) Modelos dinámicos: Enfatizan la cualidad conductual de los sistemas. "Dinámico" puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.
4) Modelos de proceso: Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento (script) de proceso. Esta vista se ejemplifica con el actual trabajo sobre programación de procesos para derivar arquitecturas.
5) Modelos funcionales: Una minoría considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un framework particular.

Cada paradigma de desarrollo exige diferente número y tipo de vistas o modelos para describir una arquitectura. No obstante, existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

- La visión **estática**: describe qué componentes tiene la arquitectura.
- La visión **funcional:** describe qué hace cada componente.
- La visión **dinámica:** describe cómo se comportan los componentes a lo largo del tiempo y como interactúan entre sí.

Las vistas o modelos de una arquitectura pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes tales como los diagramas de estado, los diagramas de flujo de datos, etc. Estos lenguajes son apropiados únicamente para un modelo o vista.

Afortunadamente existe cierto consenso en adoptar UML (*Unified Modeling Language*, lenguaje unificado de modelado) como lenguaje único para todos los modelos o vistas. Sin embargo, un lenguaje generalista corre el peligro de no ser capaz de describir determinadas restricciones de un sistema de información (o expresarlas de manera incomprensible).

Quizá uno de los modelos más conocidos es el "4+1" de Philippe Kruchten, vinculado al Rational Unified Process (RUP), que define cuatro vistas diferentes (ver figura 6):

- Vista de casos de uso o de escenarios: A través de esta, los arquitectos pueden desarrollar las siguientes cuatro vistas.
- Vista lógica: Describe el modelo de objetos.

- Vista de proceso: Muestra la concurrencia y sincronía de los procesos.
- Vista física: Muestra la ubicación del software en el hardware.
- Vista de Implementación: Describe la organización del entorno de desarrollo.

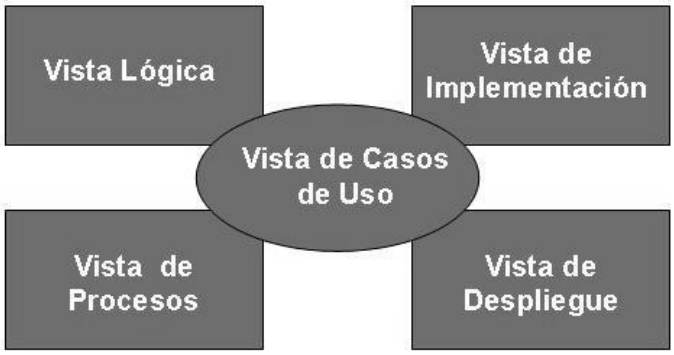


Figura 6: Vistas de la arquitectura propuesta por RUP.

1.6 Patrones Arquitectónicos

Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. **(19)**

Los patrones expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos; así como ayudan a especificar la estructura fundamental de una aplicación.

1.6.1 Patrón en Capas

Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Instrumentan así una vieja idea de organización estratigráfica que se remonta a las concepciones formuladas por el patriarca Edsger Dijkstra en la década de 1960, largamente explotada en los años subsiguientes. En algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes, y excepto para funciones puntuales de exportación; en estos sistemas, los componentes implementan máquinas virtuales en alguna de las capas de la jerarquía. En otros sistemas, las capas pueden ser sólo parcialmente opacas. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. El uso de arquitecturas en capas, explícitas o implícitas, es muy frecuente. **(20)**

En un patrón en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción. Los diagramas de sistemas clásicos en capas dibujaban las capas en adyacencia, sin conectores, flechas ni interfaces; en algunos casos se suele representar la naturaleza jerárquica del sistema en forma de círculos concéntricos. Las restricciones topológicas del patrón pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros elementos de la misma; se supone que si esta exigencia se relaja, el patrón deja de ser puro y pierde algo de su capacidad heurística **(21)**; también se pierde, naturalmente, la posibilidad de reemplazar de cuajo una capa sin afectar a las restantes, disminuye la flexibilidad del conjunto y se complica su mantenimiento. Las formas más rígidas no admiten ni siquiera pass- through: cada capa debe hacer algo, siempre. En la literatura especializada hay multitud de argumentos a favor y en contra del rigor de esta clase de prescripciones. A veces se argumenta que el cruce superfluo de muchos niveles involucra eventuales degradaciones de performance; pero muchas más veces se sacrifica la pureza de la arquitectura en capas precisamente para mejorarla: colocando, por ejemplo, reglas de negocios en los procedimientos almacenados de las bases de datos, o articulando instrucciones de consulta en la capa de la interfaz del usuario.

Casos representativos de este patrón son muchos de los protocolos de comunicación en capas. En ellos cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más bajos suelen estar asociados con conexiones de hardware.

Ventajas y Desventajas del Patrón en Capas

Las ventajas del patrón en capas son obvias. Primero que nada, el patrón soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el patrón admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas.

También se han señalado algunas desventajas de este patrón. **(21)** Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de performance pueden requerir acoplamientos específicos entre capas de

alto y bajo nivel. A veces es también extremadamente difícil encontrar el nivel de abstracción correcto; por ejemplo, la comunidad de comunicación ha encontrado complejo mapear los protocolos existentes en el framework ISO, de modo que muchos protocolos agrupan diversas capas, ocasionando que en el mercado proliferen los drivers o los servicios monolíticos. Además, los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples. **(21)**

1.6.2 Patrón Modelo Vista Controlador

La arquitectura Modelo Vista Controlador surgió como patrón arquitectónico para el desarrollo de interfaces gráficos de usuario. Su concepto se basaba en separar el modelo de datos de la aplicación de su representación de cara al usuario y de la interacción de éste con la aplicación, mediante la división de la aplicación en tres partes fundamentales, procesamiento, salida y entrada. Para esto, utiliza las siguientes abstracciones:

- **Modelo (Model):** Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.
- **Vista (View):** Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- **Controlador (Controller):** Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio (“service requests”) para el modelo o la vista.

Se han desarrollado a lo largo de los años, desde la presentación de este patrón a la comunidad científica 3 variantes fundamentales, que se presentan brevemente a continuación.

Variante I: (Figura 7)
Variante en la cual no existe ninguna comunicación entre el Modelo y la Vista y esta última recibe los datos a mostrar a través del Controlador.

Variante II: (Figura 8)
Variante en la cual se desarrolla una comunicación entre el Modelo y la Vista, donde esta última al mostrar los datos la busca directamente en el Modelo, dada una indicación del Controlador, disminuyendo el conjunto de responsabilidades de este último.

Variante III: (Figura 9)

Variante en la cual se diversifica las funcionalidades del Modelo teniendo en cuenta las características de las aplicaciones multimedia, donde tienen un gran peso las medias utilizadas en estas.

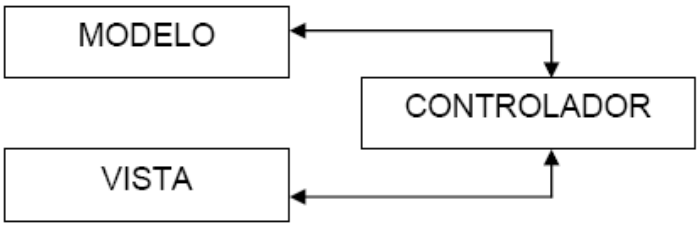


Figura 7: Variante inicial del Patrón MVC.

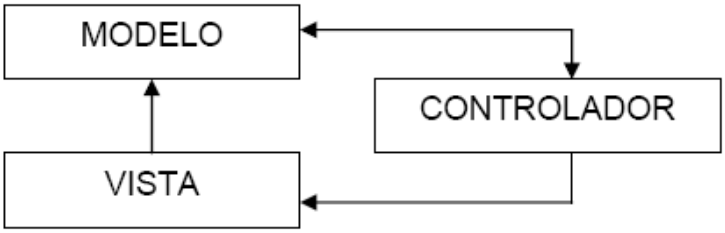


Figura 8: Variante Intermedia del Patrón MVC.

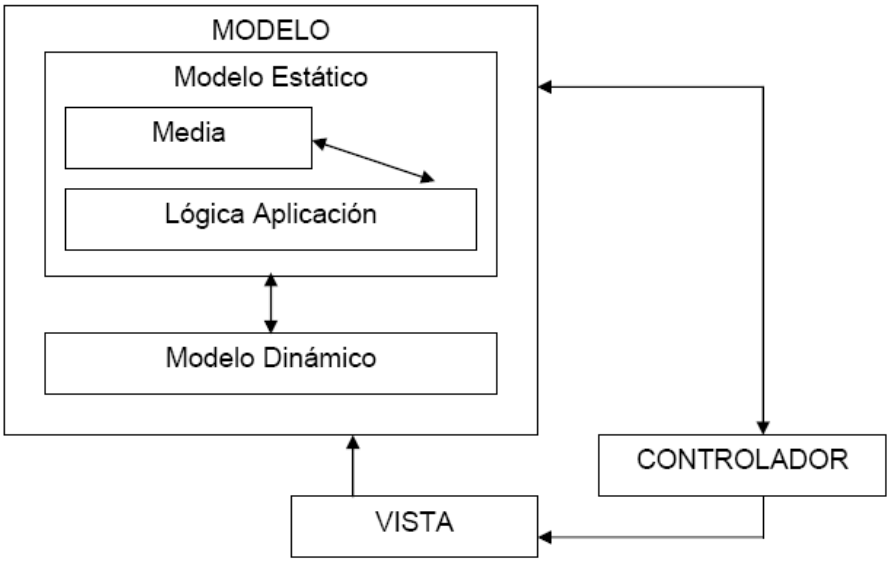


Figura 9: Variante modificada para Aplicaciones Multimedia del MVC, conocida como MVCMM. (Tomado de (22))

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario. Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (si se utiliza la variante 2 descrita anteriormente, de lo contrario lo obtiene a través del Controlador). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. *Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista, según lo descrito en la segunda variante.*
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente

Algunos de sus principales beneficios son:

- Menor acoplamiento.
- Mayor cohesión.
- Las vistas proveen mayor flexibilidad y agilidad.
- Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales.
- Más claridad de diseño.
- Facilita el mantenimiento.
- Mayor escalabilidad.

1.7 Una variación que acerca el patrón MVC al software educativo cubano.

Para poder comprender lo que es el Software (y consecuentemente la ingeniería del Software), es importante examinar las características del Software que lo diferencian de otras cosas que los hombres pueden construir.

El Software es un elemento del sistema que es lógico, en lugar de físico. Por tanto el software tiene unas características considerablemente distintas a la del hardware.

El software educativo cubano, tiene diferentes características marcadas en comparación con los desarrollados en otros países del mundo. Todo esto producido por el avance y la experiencia acumulada en el área de la pedagogía en el país que lo ubica en uno de los primeros lugares en el planeta en este sentido.

Las aplicaciones educativas cubanas explotan grandemente los conceptos del entorno hipermedia, así como incorporan de forma profunda las técnicas y conocimientos de bases de datos relacionales y la Programación Orientada a Objetos más avanzada que se utiliza hoy día; como consecuencia de la necesidad de implementar diversos y complejos métodos pedagógicos desarrollados por nuestros educadores en las distintas ramas de la enseñanza cubana.

Si en otros países las multimedia llamadas educativas se circunscriben a meros software que presentan y evalúan alguna determinada materia, en Cuba por el contrario se desarrollan productos que realizan análisis exhaustivos de la navegación del usuario, mantienen actualizado el historial de trabajo de los usuarios y permiten preparar el software para diferentes entornos de trabajo a dichos usuarios teniendo en cuenta determinadas características. Esto hace que nuestras aplicaciones se vean grandemente recargadas en el almacenamiento, procesamiento y actualización de una gran cantidad de información constantemente.

Durante las dos pasadas décadas, se han desarrollado un gran número de métodos de modelado. Los investigadores han identificado los problemas del análisis y sus causas y han desarrollado varias notaciones de modelado y sus correspondientes conjuntos de heurísticas para solucionarlos (…) Se emplean modelos para poder comunicar de forma compacta las características de la función y su comportamiento. Se aplica la partición para reducir la complejidad. Son necesarias las visiones esenciales y de implementación del software para acomodar las restricciones lógicas impuestas por los requisitos del procesamiento y las restricciones físicas impuestas por otros elementos del sistema. **(14)**

Precisamente por las características del software educativo cubano, los diseñadores han buscado y aplicado soluciones ya trabajadas internacionalmente y con un gran número de aplicaciones prácticas en este sentido, dentro de las que se encuentra la utilización del Patrón MVC. No obstante este patrón

tal y como está descrito actualmente, aún con su variante más actual no responde completamente a las características mencionadas.

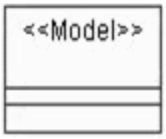
El diseño del software, al igual que los enfoques de diseño de ingeniería en otras disciplinas, va cambiando continuamente a medida que se desarrollan métodos nuevos, análisis mejores y se amplía el conocimiento.

El Software se construye para procesar los datos, para transformar datos de una forma u otra, es decir, para aceptar una entrada de información, manipularla de alguna manera y producir una salida de información (…) es importante recalcar, sin embargo, que el software también procesa sucesos. Un suceso representa algún aspecto de control del sistema y no es más que un dato binario (es encendido o apagado, verdadero o falso, está allí o no). **(14)**

Como ya ha sido descrito en el epígrafe anterior el MVC divide la arquitectura de una aplicación en 3 tipos de clases fundamentales, con las responsabilidades siguientes:

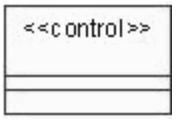
Clase Vista <<View>>:

- Recepcionar peticiones.
- Mostrar respuestas del sistema.



Clase Modelo <<Model>>:

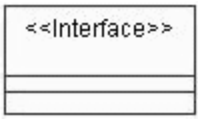
- Procesar peticiones del sistema.
- Generar datos de respuesta del sistema.
- Gestionar y almacenar la información.



Clase Controlador <<Controller>>:

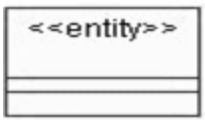
- Gestionar procesamiento de peticiones.
- Gestionar muestra de respuestas del sistema.

De igual forma si analizamos el *Lenguaje Unificado de Modelado (UML)* [Unified Modeling Language], este propone para el desarrollo del Modelo de Análisis de las aplicaciones, tres tipos de clases fundamentales, con las cuales podemos expresar todas las funciones de cualquier software, con sus respectivas responsabilidades:



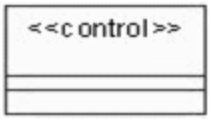
Clase Interfaz <<Interface>>:

- Recepcionar peticiones al sistema.
- Mostrar respuestas del sistema.
-



Clase Entidad <<Entity>>:

- Gestionar datos (información) necesaria para el sistema.
- Almacenar datos (información) persistentes del sistema.



Clase Controlador <<Controller>>:

- Procesar Información del sistema.
- Gestionar visualización de respuesta del sistema.

Las responsabilidades se relacionan con las obligaciones de un objeto respecto a su comportamiento. Esas responsabilidades pertenecen, esencialmente, a las dos categorías siguientes:

1. *Conocer*
2. *Hacer*

Entre las responsabilidades de un objeto relacionadas con *hacer* se encuentran:

- Hacer algo en uno mismo
- Iniciar una acción en otros objetos
- Controlar y coordinar actividades en otros objetos

Entre las responsabilidades de un objeto relacionadas con *conocer* se encuentran:

- Estar enterado de los datos privados encapsulados
- Estar enterado de la existencia de objetos conexos
- Estar enterado de cosas que se puede derivar o calcular

"La calidad de diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener y entender, reutilizar o extender. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos." **(23)**

Precisamente en lo expresado por Craig Larman referente a la asignación de responsabilidades a las clases, basamos el análisis del diseño de aplicaciones educativas en Cuba, las cuales refuerzan o sobrecargan a determinadas clases al ser responsables de soportar la mayoría de las acciones en este tipo de software.

Se presentan a continuación un conjunto de consideraciones al respecto:

1. Según lo planteado en UML, la clase Controlador queda sobrecargada en sus responsabilidades al asumir el procesamiento de toda la información necesaria en el sistema.
2. La clase Entidad, utilizando los conceptos de UML, separa eficientemente de la clase Controladora la responsabilidad referente al almacenamiento y gestión de la información persistente en la aplicación, lo que es sumamente beneficioso, no solo desde el punto de vista del diseño, sino para la implementación y mantenimiento del sistema.

3. La clase Modelo, según el Patrón MVC, asume todas las responsabilidades que UML deposita en la clases Entidad, sobrecargando (dadas las características del Software Educativo Cubano y su gran trabajo con las Bases de Datos) dicha clase por completo.

Por estas consideraciones realizadas, se propone las siguientes modificaciones, las cuales se muestran gráficamente en los Anexos 1 y 2:

1. Mantener la clase Vista (Interfaz según UML) del patrón original con sus responsabilidades establecidas.
2. Mantener la separación que MVC establece entre las clases Modelo y Controlador (dividiendo las responsabilidades que UML asigna a la clase Controlador), quitándole a la Clase Modelo la responsabilidad asociada con la gestión y almacenamiento de los datos persistentes en la aplicación.
3. Definir una nueva clase denominada Modelo – Entidad, que asuma la responsabilidad de la gestión y el almacenamiento de toda la información persistente de la aplicación, así como la comunicación con la Base de datos del sistema en cuestión.

1.8 Análisis de otras soluciones ya existentes

En el mundo actualmente muchas son las organizaciones o grupos de investigadores que trabajan en la producción de software educativo, un paso importante para el buen funcionamiento de este tipo de aplicaciones es la definición de una adecuada arquitectura.

El departamento de Historia de América II: Antropología de América, conjuntamente con el de programación de la Universidad Complutense de Madrid, tiene como objetivo convertir en recursos educativos los materiales arqueológicos y etnográficos disponibles en el primero de los citados departamentos. Para construir estos recursos educativos se ha desarrollado el concepto de *objeto virtual*: un objeto digital que sirve para agrupar toda la información relacionada con un determinado objeto arqueológico o etnográfico. Posteriormente, y a partir de los objetos virtuales básicos, se construyen otros recursos educativos más elaborados, que se integran, junto con los objetos virtuales, en un entorno Web, que puede utilizarse tanto para la enseñanza como para la investigación y divulgación arqueológicas.

Para resolver estos problemas decidieron utilizar una arquitectura basada en el modelo MVC (*Model-View- Controller*). **(24)** Este modelo divide los módulos de la aplicación en tres categorías o capas:

- ***Modelo*.** Contiene la funcionalidad y el estado de la aplicación. En él se define la estructura de los datos relevantes al dominio y se definen las funciones que procesan esos datos. En esta parte se integra el acceso, recuperación y modificación de la información relativa a los objetos virtuales, así como a la meta-información asociada a cada objeto; de esta forma se logra un máximo de independencia entre el modelo y el resto de la aplicación.
- ***Vista*.** Proporciona el interfaz del modelo y los mecanismos de interacción con el usuario; la vista puede acceder al estado del modelo, pero no puede modificarlo; hay que "avisar" a la vista cuando se produzca algún cambio en el modelo. En la vista están los módulos que se procesan en el lado del cliente: HTML, Applets y JavaScript
- ***Controlador*.** El controlador establece la conexión entre los elementos del interfaz y los datos que estos representan. El controlador implementa el flujo de la aplicación, se ejecuta en el servidor y depende de la situación actual del modelo, así como de las acciones realizadas por el usuario sobre la vista.

Para poder usar este modelo en un entorno Web, se tienen que hacer algunas modificaciones. La razón fundamental es que en un servidor HTTP no se pueden notificar los cambios en el modelo a la vista para que ésta se actualice, ya que es el navegador quien debe hacer una nueva petición al servidor (e.g. petición de recarga de la página), para comprobar si ha cambiado el estado del modelo. También hay que tener en cuenta que en general la vista usa distintas tecnologías (HTML/JavaScript) que el modelo-controlador (en este caso PHP).

Esto lleva a la definición de un modelo MVC específico para la red conocida como modelo MVC 2 **(25)**, que tiene la estructura descrita en la Fig. 10. En este esquema cuando el usuario realiza una acción sobre el interfaz en la parte del cliente (navegador Web) que origina una nueva petición al servidor, ésta es procesada por el controlador. El controlador, en función de la petición que le llega, puede realizar alguna modificación sobre el modelo de datos, y redireccionar a la página adecuada; desde esta página se puede realizar alguna consulta sobre el estado del modelo de datos para completar la respuesta, y el resultado se envía al cliente; de esta forma se cierra el ciclo de operación.

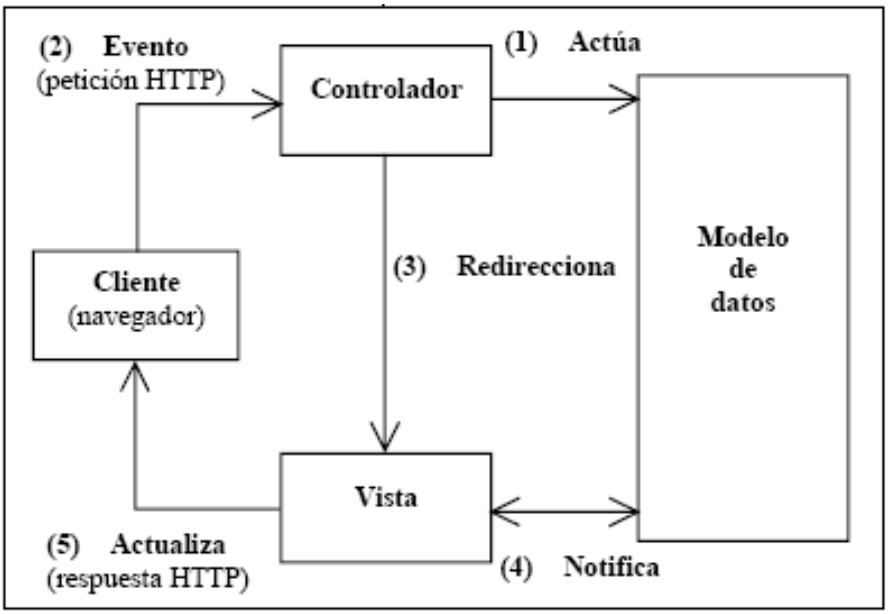


Figura 10: Modelo-Vista-Controlador 2.

En Cuba, específicamente en la UCI se han desarrollado muchas multimedias educativas tales como: "Curso Optativo Interactivo Flash Básico". En esta aplicación bajo el paradigma orientado a objeto se utiliza el patrón MVC para la interfaz de usuario. El paradigma MVC es un modelo de arquitectura conocido en el desarrollo de aplicaciones orientadas a objetos que distinguen un componente *modelo*, un componente *vista* para mostrar la información al usuario y un componente *controlador* para manipular los eventos de interacción.

1.9 Los Arquitectos de Software

A diferencia de un programador, el arquitecto de software debe dominar la mayor cantidad de tecnologías de software y prácticas de diseño, para así poder tomar decisiones adecuadas para garantizar el mejor desempeño, reusabilidad, robustez, portabilidad, flexibilidad, escalabilidad y mantenibilidad de las aplicaciones.

El arquitecto de software es el líder técnico del equipo, el rol natural al que debe aspirar un programador experimentado que desea tomar decisiones técnicas relevantes en el desarrollo de un sistema. Es el principal tomador de decisiones respecto a la manera en que será construida la aplicación por los programadores del equipo. El líder de proyecto se apoya totalmente en este rol para

alcanzar el éxito del proyecto optimizando el uso de la tecnología para desarrollar la solución correcta que proporcionará valor real a sus usuarios y al negocio al que le dará soporte.**(26)**

Hay dos formas de convertirse en arquitecto: aprendiendo a definir las soluciones con base en la propia experiencia, o reutilizando el conocimiento de los expertos a nivel mundial plasmado en patrones de arquitectura y diseño.

El arquitecto es el líder técnico del proyecto y tiene que estar al lado de los programadores velando que las cosas se hagan de acuerdo como se ha especificado en la arquitectura, es importante también su presencia para ayudar y guiar a los desarrolladores novatos que aún no conocen la arquitectura.

CAPÍTULO 2: DESCRIPCIÓN DE LA PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se hará un análisis de cómo se desarrolla el software educativo en la UCI, se van a identificar los elementos estructurales comunes y significativos utilizados en éste, además se realizará una propuesta de los elementos estructurales y lógicos a utilizar en la producción de software educativo en la UCI, para darle cumplimiento al objetivo general de la tesis.

2.2 Proceso de desarrollo del Software Educativo en la UCI

El desarrollo de la Industria Cubana del Software se encuentra estrechamente relacionado a la UCI, la misma constituye un nuevo modelo formación – investigación – producción en el campo de las TIC y su impacto se hace sentir ya en diferentes sectores de la sociedad y la economía nacional.
La estrategia adoptada en la producción de los productos de software educativo y multimedia en la UCI para garantizar la adecuada ejecución de los proyectos consta de un flujo general de trabajo que se divide en seis procesos, una imagen gráfica del flujo se puede encontrar en el anexo 3:

- Definición del proyecto.
- Gestión de requisitos y análisis.
- Evaluación técnica.
- Gestión de medias.
- Gestión de diseño gráfico.
- Construcción.

El primer proceso es la definición del proyecto, etapa de concepción inicial del producto, donde el cliente entrega su solicitud y el guión de contenido (si lo tiene elaborado) y se define la estrategia de trabajo.
El proceso de Gestión de requisitos y análisis resulta de vital importancia ya que aquí se realiza una descripción detallada del objeto de estudio, se analiza la solicitud del cliente y del guión de contenido y se aclaran las posibles dudas sobre la documentación entregada. Se revisa el guión hasta tener una total comprensión del mismo y cuando se logra esto se elabora el guión técnico. Debe quedar clara la

necesidad de elaborar el producto, el público al que va dirigido, los objetivos pedagógicos que se pretenden cumplir, los contenidos a tratar y los medios para presentarlos.

En el proceso Evaluación Técnica se realiza un análisis del proyecto en cuanto a las herramientas que se utilizarán para el desarrollo, los equipos de trabajo que se conformarán, el hardware necesario tanto para realizadores como para usuarios, la factibilidad técnica y económica de su producción (presupuesto necesario), las formas de distribución, las especificaciones de los productos a elaborar y se realiza una recomendación de la arquitectura a usar para la producción de software educativo así como de la arquitectura organizativa para acometer la producción.
La gestión de medias comprende los procesos de búsqueda o producción de determinados recursos audiovisuales en función de los requisitos dados en el guión técnico. Esta tarea es responsabilidad del equipo de audiovisuales de la Dirección de Comunicación Audiovisual.
La gestión de diseño gráfico comprende los procesos de concepción y realización del diseño gráfico de un producto de software educativo en función de los requisitos dados en el guión técnico. En esta etapa se obtendrá una información detallada de cómo estará estructurado el programa y el diseño de la interfaz de cada una de las pantallas por parte del equipo de diseñadores de la Dirección de Diseño Visual.
En el proceso de Construcción se cumplen dos tareas de singular importancia: la obtención y edición de todos los medios que serán empleados y la programación por parte de los proyectos en cada facultad. La implementación se rige por el guión técnico, donde se realizan revisiones técnicas para corregir posibles errores antes de ser entregado al cliente.

2.3 Multimedia Actual

Hoy en día, los cambios augurados son una realidad y los multimedios son tan comunes que resulta impensable una computadora sin ellos. Los multimedios computarizados emplean los medios - la palabra (hablada y escrita), los recursos de audio, las imágenes fijas y las imágenes en movimiento- para tener una mayor interacción con el usuario quien ha pasado de ser considerado como alguien que esporádicamente empleaba una computadora (con el respectivo recelo e inseguridad) a ser quien la maneja como una herramienta más en su beneficio (con ideas más claras y exigencias nuevas).

Las aplicaciones multimedia comprenden productos y servicios que van desde la computadora (y sus dispositivos "especiales" para las tareas multimedia, como bocinas, pantallas de alta definición, etc.)

donde se puede leer desde un disco compacto hasta las comunicaciones virtuales que posibilita Internet, pasando por los servicios de vídeo interactivo en un televisor y las videoconferencias.

Teniendo claro este panorama, pasemos a ver cómo los programas multimedia pueden satisfacer estas necesidades. Veamos algunas de las fortalezas de estos programas que se convierten en ventajas con respecto a sistemas tradicionales.

En primer término está la capacidad de comunicación. Por su misma definición, los programas multimedia tienen la capacidad de utilizar diferentes medios para comunicar ideas. Textos, gráficas, sonidos, videos y animaciones, interactuando armónicamente, pueden lograr en pocos minutos transmitirle a la audiencia toda la información necesaria, por voluminosa que ésta sea. Si a esto le agregamos la interactividad, que es la capacidad que tienen estos programas para permitirles a los usuarios "navegar" por la información en el orden y velocidad que deseen, obtendremos el impacto necesario para nuestra labor de mercadeo. Claramente, este punto se convierte en una ventaja frente a los medios tradicionales.

Otra ventaja importante es la flexibilidad. Esta ventaja no es exclusiva de los programas multimedia, sino en general de los programas de computador. La mayoría de herramientas para desarrollo de programas multimedia permiten la utilización de metodologías como programación orientada por objetos, que aceleran la construcción de las aplicaciones y permiten la reutilización de código ya existente. Adicionalmente, la utilización de bases de datos y el desarrollo escalar o por etapas, permiten que los programas multimedia tengan una fácil actualización y por consiguiente gran agilidad para evolucionar y adaptarse a los cambios.

Por último, así como el costo es uno de los principales problemas para el mercadeo, se constituye en una de las grandes oportunidades para los sistemas multimedia. Esto nos introduce al tema de los medios de difusión, porque es allí donde se hace más notoria esta ventaja.

Los programas multimedia por su alto contenido de información, deben ser distribuidos en medios de gran capacidad, que hagan práctica su utilización. El CD-ROM se ajusta a estas características y además, hoy por hoy, es un medio muy económico. Por estas razones, se ha convertido en el medio por excelencia para distribuir multimedia, tanto así que hoy, un computador sin CD-ROM no se considera un computador multimedia, así tenga capacidades para ejecutar video y sonido.

Los actores que intervienen en general en la producción de un sistema u obra multimedia son entre otros los siguientes protagonistas:

- El productor: Es el director general del proyecto
- Los expertos en el tema o contenido: Son los especialistas sobre una temática particular. Son los dueños del "conocimiento".
- El pedagogo: Es el experto en poder transmitir en forma coherente y utilizando todos los medios los conocimientos del experto.
- El guionista: Es el especialista encargado de "volcar" en escenas específicas las ideas del experto y el método del pedagogo.
- Los diseñadores: Son los expertos en diseño gráfico.
- Productores de objetos: Son los dibujantes, fotógrafos, productores de video, animadores, etc.
- Los programadores: Son expertos en la programación de lenguajes autores.

2.3.1 Multimedia en la Educación

La informática encontró una buena vía de acceso a los hogares y fue por medio de la multimedia. Esta evolución en los computadores domésticos, ha hecho lo que hasta hace un par de años era una aburrida máquina de proceso de texto y archivo de datos se haya convertido en una excelente máquina con capacidad de mostrar video y sonido al mismo tiempo y con calidad de compact disk en un mismo aparato.

La multimedia también sirve como un medio educativo, cultural para los niños; actualmente existen colegios tanto primarios como secundarios que utilizan computadores como un medio de enseñanza y aprendizaje; ya sea tanto teórica como práctica; y para estos utilizan software que abarcan diversos temas, que comprenden desde la matemática, geografía, ciencia, artística, gramática y hasta inclusive música con ellos.

Los profesores se han dado cuenta de las grandes posibilidades que los CD-ROMs brindan en materia educativa: son obras cada día más completas que motivan por sus grandes números de estímulos, el aprendizaje.

2.3.2 Componentes de una obra multimedia

- tipos de contenido: Texto, Sonido, Gráficos, Animación, Video.
- Diseño.
- Escenas (Pantallas interactivas).
- Sistemas de navegación, instrumentos de navegación.

2.4 Elementos estructurales comunes en la producción del Software Educativo en la UCI

El 26 de febrero del 2003, surge el Departamento Nacional de Software Educativo, por la necesidad de ampliar y profundizar desde el punto de vista científico y organizativo la evaluación del desarrollo de los software educativos en todo el país, lo que permitió dar una respuesta inmediata a las profundas transformaciones educacionales que se estaban llevando a cabo en el fragor de la "Batalla de ideas".

Unos de los objetivos propuestos por esta institución, es: coordinar con otras instituciones, en particular con la Universidad de las Ciencias de la Informática (UCI), las acciones referentes a la industria del software educativo.

La UCI como Universidad de excelencia en un ámbito productivo y educativo, se ha estructurado en un conjunto de 10 facultades docentes y una Infraestructura Productiva (IP) con un conjunto de direcciones, dentro de las cuales se encuentra la Dirección de Producción de Software Educativo, que en conjunto con las facultades 5, 8 y 9 desarrolla en la actualidad productos destinados a la educación en diferentes sectores de la sociedad cubana y la exportación.

Uno de los principales objetivos de la Dirección de Producción de software educativo, es el desarrollo de multimedia educativas. Hasta ahora este trabajo se realiza manualmente, esto implica comenzar a desarrollar cada software y realizar su implementación a partir de cero a veces sin el conocimiento necesario para ello, lo que además de consumir grandes recursos humanos y de tiempo, afecta la calidad de los productos que se realizan.

Estando de acuerdo con los antes planteado, cabe mencionar que hoy en la universidad se utiliza para la modelación de las aplicaciones multimedia educativas lo siguiente:

- Guiones de contenido y técnicos.
- Árboles o mapas de navegación.
- Análisis de posibles arquitecturas para los productos software, máxime cuando estos son colecciones.

2.4.1 El guión técnico y de contenido

"El guión de un multimedia en soporte informático tiene una estructura diferente, con columnas diferenciadas para imagen, sonido, texto y acciones (o interacciones). En cada una de ellas hay que identificar el recurso digital (en forma de fichero informático), así como los resultados de determinadas acciones sobre zonas específicas de la pantalla". **(27)**

"(…) el guión (…) abarca desde los aspectos estructurales y funcionales hasta los formales y estéticos, con un nivel de detalle que en principio permite desarrollar la ejecución del proyecto sin ambigüedades.

Se discute mucho hasta dónde deben especificarse los detalles, pero mientras más preciso sea el guión mucho más rápidamente y a un menor costo se podrá llevar a cabo la elaboración de la obra." **(28)**

"La función primordial del guión es dar objetividad al proyecto de la obra multimedia de forma tal que pueda independizarse el proceso de ejecución del proyecto de concepción y diseño. (…) En segundo lugar, el guión es imprescindible para lograr una comunicación clara y precisa entre los integrantes del equipo de trabajo de manera que aunque realicen tareas independientes todos conozcan cómo tienen que hacer su labor y cómo encaja cada uno dentro de la obra." **(28)**

El Guión Técnico es elaborado por el profesional informático a medida que va comprendiendo la idea del docente. Consiste en definir las bases de la realización, la metodología, los programas a utilizar, los formatos de presentación, diseño de pantalla, los efectos a utilizar en cada parte, etc.

El Guión de Contenido indica el material textual que se va a utilizar en las diferentes secuencias y la manera en la que se va relacionando mediante una jerarquización conceptual que irá de lo más importante o lo más general o específico y que deberá transmitirse en forma muy clara en el guión.

2.4.2 Árboles o mapas de navegación.

Un mapa de navegación es la representación gráfica de la organización de la información. Expresa todas las relaciones de jerarquía y secuencia y permite elaborar escenarios de comportamiento de los usuarios. También grafica, de modo que todos los profesionales participantes en un proyecto lo tengan claro, diferencias entre páginas dinámicas, administrables o estáticas.

El principal valor de un mapa de navegación es que permite anticipar errores de organización de la información, de modo de corregirlos cuando aún no se ha invertido tiempo y dinero en la construcción del producto.

2.4.3 Análisis de posibles arquitecturas para los productos de software educativos en la UCI

Antes de identificar una posible arquitectura o posibles elementos arquitectónicos a utilizar en el desarrollo del software educativo en la UCI, en este epígrafe se hará primero un análisis de los diferentes elementos estructurales comunes y significativos utilizados en el software educativo en la UCI.

Proyecto Multimedia Historia Universal versión 2.0 (UCI)

En este producto se utilizan herramientas verticales y horizontales. Las primeras son las que permiten de forma directa la realización del SWE y en la segunda en las que se apoyan para el trabajo con el mismo.

Entre las verticales se encuentran las siguientes:

- Herramientas de Programación: Macromedia Flash 8.0.
- Como tratamientos de Imágenes: Fireworks 8.0 y Photoshop CS2
- Herramienta de modelado: Rational Rose.

Entre las Horizontales se utilizan las siguientes:

- Controlador de versiones: SVN (Gforge).
- Estaciones de trabajo: 2 PC p4 Windows XP 512 RAM.

Lenguaje de programación: Action Script 2.0 y XML.

Esta multimedia está compuesta por 4 módulos: Presentación, Temas, Servicios y Biblioteca. Los casos de usos arquitectónicamente más significativos, son: CUS Presentación, CUS Módulo Temas, CUS Módulo Servicios, CUS Módulo Biblioteca.

Los paquetes en los que esta dividida esta multimedia son:

- Paquete General, contiene varios paquetes con los estilos CSS, Galería, Glosario, Música, XML, Videos.
- Paquete Tema1, contiene varios paquetes con los estilos CSS, Ventanas, XMLs, Música, Videos, Visores.
- Paquete Tema2, contiene varios paquetes con los estilos CSS, Ventanas, XMLs, Música, Videos, Visores.
- Paquete Tema3, contiene varios paquetes con los estilos CSS, Ventanas, XMLs, Música, Videos, Visores.
- Paquete Tema4, contiene varios paquetes con los estilos CSS, Ventanas, XMLs, Música, Videos, Visores.

- Paquete Tema5, contiene varios paquetes con los estilos CSS, Ventanas, XMLs, Música, Videos, Visores.

La Gestión de Medias de la multimedia (ver anexo 4).

La vista global de la multimedia se muestra en la siguiente figura:

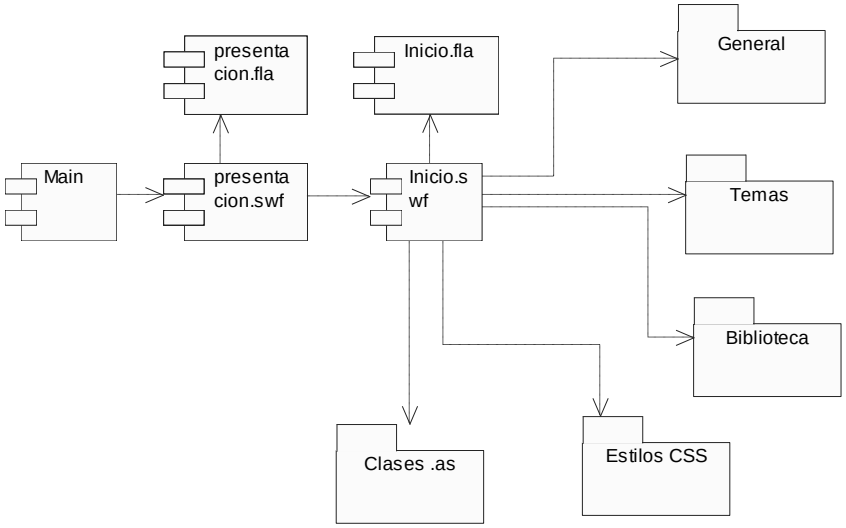


Figura 11: Diagrama de Componentes.

Proyecto MENPET (UCI)

Multimedia Revolución Energética

En este producto se utilizan herramientas verticales y horizontales.

Entre las verticales se encuentran las siguientes:

- Herramientas de Programación: Macromedia Flash 8.0.
- Herramientas de planificación: Microsoft Project.
- Como tratamientos de Imágenes: Fireworks 8.0 y Photoshop CS2.
- Herramienta de modelado: Rational Rose.

Entre las Horizontales se utilizan las siguientes:

- Controlador de versiones: SVN (Gforge).
- Estaciones de trabajo: 2 PC p4 Windows XP 512 RAM.

Lenguaje de programación: Action Script 2.0 y XML.

Esta multimedia esta compuesta por 5 módulos: Presentación, Biblioteca, Temas, Galería y Servicios. Los casos de usos arquitectónicamente más significativos, son: CUS Presentación, CUS Módulo Temas, CUS Módulo Servicios, CUS Módulo Biblioteca y CUS Módulo Galería.

Los paquetes en los que esta divida esta multimedia son:

- Paquete CSS, estos contienen las hojas de estilo, ficheros.css, etc.
- Paquete Recursos, estos contienen Animaciones, Sonidos, Videos, Imágenes y Textos.
    - Dentro este paquete también se encuentran un paquete con las Clases y otro con los Pasatiempos.
- Paquete Módulos, estos contienen los módulos Presentación, Biblioteca, Temas, Galería y Servicios.
- Paquete XML, estos contienen los contenidos, o sea los datos.

La Gestión de Medias de la multimedia (ver anexo 4).

La vista global de la multimedia se muestra en la siguiente figura:

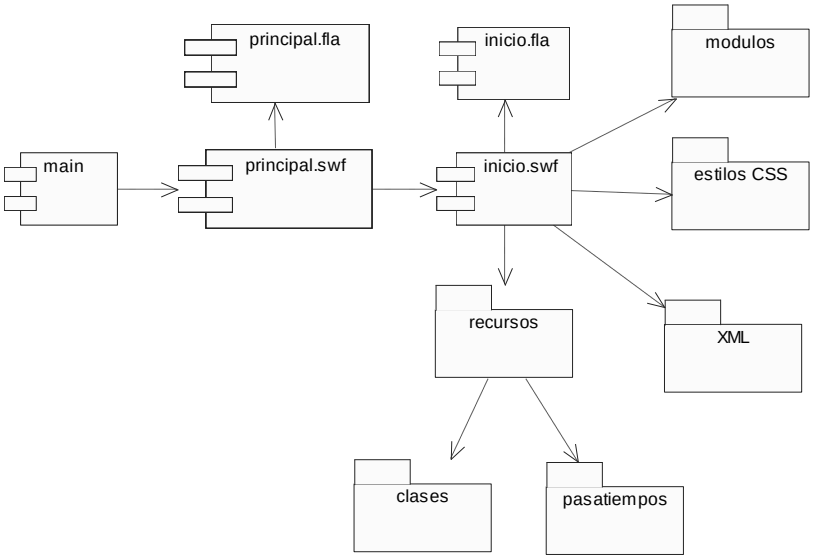


Figura 12: Diagrama de Componentes.

Proyecto Multimedia Libros Electrónicos (UCI)

En este producto se utilizan herramientas verticales y horizontales.

Entre las verticales se encuentran las siguientes:

- Herramientas de Programación: Macromedia Dreamweaver 8.0.
- Como tratamientos de Imágenes: Fireworks 8.0 y Photoshop CS2
- Herramienta de modelado: Rational Rose.

Entre las Horizontales se utilizan las siguientes:

- Controlador de versiones: SVN (Gforge).
- Estaciones de trabajo: 6 PC p4 Windows XP 512 RAM.

Lenguaje de programación: HTML.

Los paquetes en los que esta divida esta multimedia son:

- Paquete contenido: contiene páginas estáticas de HTML con la información y los vínculos hacia otras páginas HTML.
- Paquete estilos: contiene los estilos CSS y las hojas de estilos.
- Paquete Imágenes: contiene las imágenes utilizadas en el software.
- Paquete library: contiene la librería para guardar los menús.
- Paquete recursos: contiene las páginas de ayuda y las páginas con los créditos.
- Paquete template: es donde se guardan las planillas del sitio.

2.4.4 Posibles Arquitecturas a Usar

Patrón en Capas

En este tipo de arquitectura el reparto de responsabilidades es, en primera instancia, lógico. Las tres capas son:

- **Presentación,** que recibe eventos del usuario a través de la interfaz presentada (de ahí el nombre), y también formatea los resultados a desplegar (digo desplegar para no repetir *presentar*... de ahí el nombre de nuevo).
- **Negocio,** o también **Lógica de Dominio,** el dominio del problema de negocio por el cual tuvimos que hacer esta aplicación.
- **Acceso a Datos,** en esta capa simplemente hay lógica que lleva y trae información entre la capa de negocio y los repositorios o sistemas externos donde los datos se almacenan. Acá tenemos conectores, pools de conexiones, cachés, etc.

Las motivaciones para aplicar este patrón arquitectónico se suelen fundamentar en *Mantenibilidad* y *Reusabilidad*. *Mantenibilidad* porque, al emplearse las APIs (Application Programming Interface) en una forma cohesiva: las APIs de interfaz de usuario en la capa de presentación, las de manipulación de XML y bases de datos en la de acceso a datos, etc., esto permite dividir mejor los equipos de trabajo según dominio de dichas APIs. Y *Reusabilidad* (si el día de mañana queremos cambiar, por ejemplo, de repositorio de datos, seguramente el impacto se concentra en la capa de acceso a datos pero no en las otras dos).

Patrón MVC

La arquitectura basada en el modelo MVC (*Model-View- Controller*) divide los módulos de la aplicación en tres categorías o capas:

- **Modelo**. Contiene la funcionalidad y el estado de la aplicación. En él se define la estructura de los datos relevantes al dominio y se definen las funciones que procesan esos datos.
- **Vista**. Proporciona el interfaz del modelo y los mecanismos de interacción con el usuario; la vista puede acceder al estado del modelo, pero no puede modificarlo; hay que "avisar" a la vista cuando se produzca algún cambio en el modelo.
- **Controlador**. El controlador establece la conexión entre los elementos del interfaz y los datos que estos representan.

Entre las ventajas que se suele tener al aplicar este patrón arquitectónico esta que:

- Al separar de manera clara la lógica de negocio (modelo) de la vista permite la reusabilidad del modelo, de modo que la misma implementación de la lógica de negocio que maneja una aplicación pueda ser usado en otras aplicaciones, sean éstas Web o no.
- Permite una sencilla división de roles, dejando que sean diseñadores gráficos sin conocimientos de programación o desarrollo de aplicaciones los que se encarguen de la realización de la capa vista, sin necesidad de mezclar código Java entre el código visual.

2.5 Definición de la propuesta de la arquitectura a utilizar en la producción de software educativo en la UCI

Después de un análisis de los elementos estructurales que se utilizan en la producción de software educativo en la UCI, se propone la utilización del Patrón en 3 Capas como arquitectura para el desarrollo de los mismos.

Componentes o elementos

Las principales componentes que distinguen la arquitectura en 3 capas son:

- **Capa Presentación:** es la capa encargada de recibir eventos del usuario a través de la interfaz, va a estar compuesta por las *vistas*, *útiles* y la *navegación*. Esta capa va a contener

por cada módulo de la aplicación, las vistas, con los estilos CSS y útiles necesarios para su desarrollo. En caso de ser una aplicación en web, cada vista sería una página en HTML.

- **Capa Lógica:** es la capa en la cual se encuentra la lógica de dominio por el cual se realiza la aplicación. Esta capa va a manejar las *clases* del negocio que se vayan a utilizar en la aplicación, además va a tener *útiles* y *servicios*, estos últimos van a ser de impresión, sonido, búsqueda, en general, los que determine el usuario. Dicha capa va a contener por cada módulo de la aplicación las clases que necesite y los útiles y servicios requeridos para su funcionamiento.
- **Capa de Acceso a Datos:** es donde residen los datos y es la encargada de acceder a los datos. Reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio. Esta capa contiene los *XML*, que son los ficheros que guardan los datos, en el caso de ser una aplicación Web se podría utilizar, en dependencia de la complejidad, una base de datos, además contienen *útiles* y los *recursos* que serían las animaciones, sonidos, videos, imágenes y textos. Dicha capa va a contener por cada módulo de la aplicación los XML, recursos y útiles necesarios para su funcionamiento.

La representación grafica de la Arquitectura 3 Capas propuesta es la siguiente:

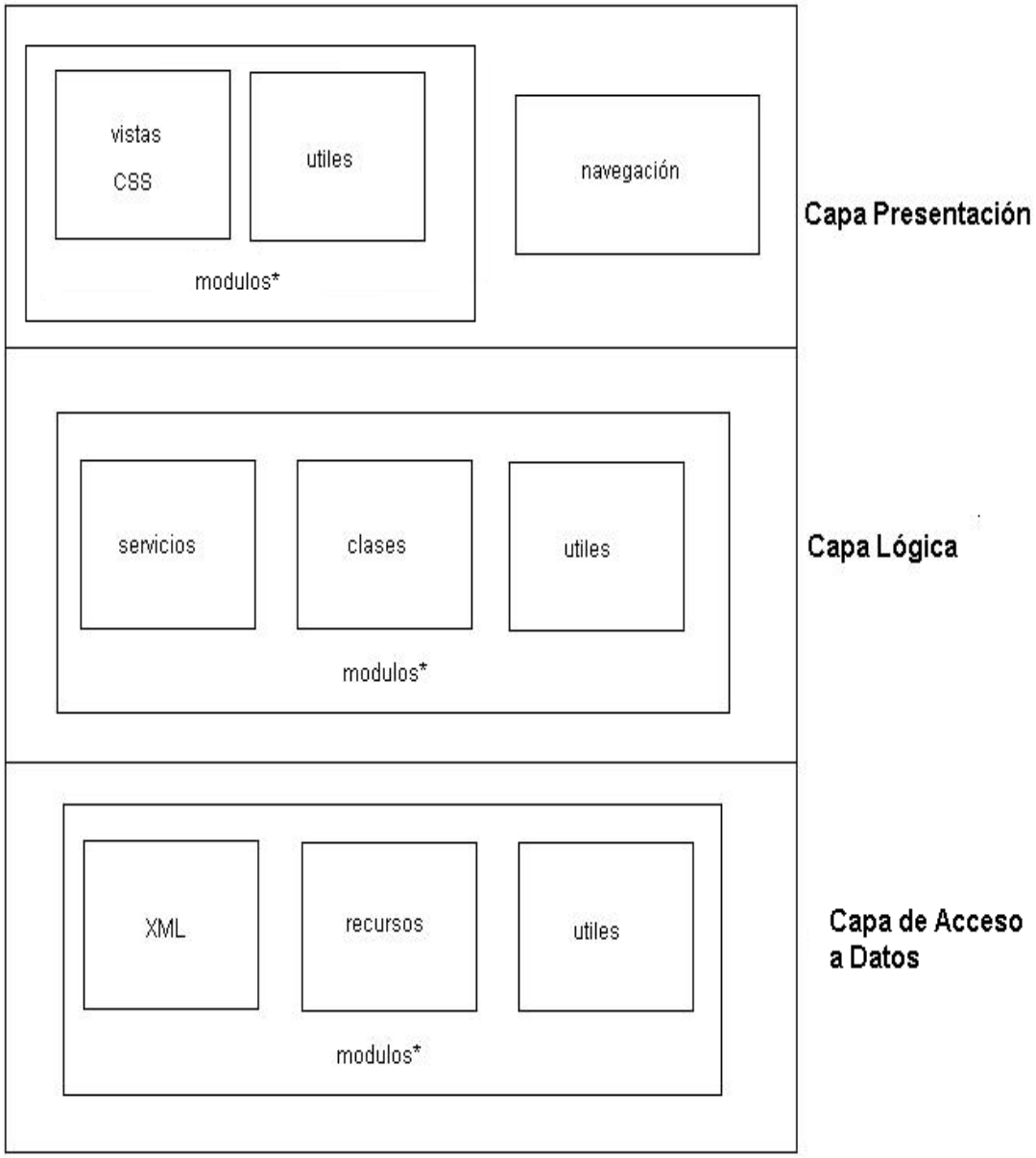


Figura 13: Representación gráfica de la Arquitectura 3 Capas Propuesta.

Para un mejor entendimiento los accesorios ***útiles*** son utilerías indispensables que debe contener cualquier herramienta multimedia para desarrollar algunas tareas, peculiares pero repetidas con frecuencia. Estos son los accesorios conformables y bien empleados que hacen más fácil su vida con la PC. Estos útiles van a contener por lo general particularidades propias de la aplicación en sí, como

también código, interfaces y clases reutilizables. Además ***módulo**** significa que se repite lo que contiene esa capa por cada módulo de la aplicación. La ***navegación*** va a ser el flash principal o en caso de una aplicación web, una pagina principal en HTML.

El **flujo de datos** de la siguiente propuesta se define como: la capa presentación al interactuar con el usuario, recibe una solicitud del mismo y le manda dicha solicitud a la capa lógica, la cual es la encargada de manejar las solicitudes y en general la aplicación; la capa lógica le pide la información necesaria a la capa de acceso a datos, ya que esta contiene los datos y en general la información de la aplicación; la capa de acceso a datos le envía la información a la capa lógica y a su vez está maneja la información y se la envía a la capa presentación, siendo está última la encargada de mostrarle la información y/o los datos al usuario.

El estilo de capas facilita la modularidad del sistema, la localización de errores y mejora considerablemente el soporte del mismo, cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Además al tener las capas separadas tenemos que existe poco acoplamiento entre las mismas, de modo que es mucho más fácil hacer modificaciones en ellas sin que interfieran en las demás.

Sus principales **ventajas** son:

- Permite el desarrollo paralelo del sistema por cada capa.
- Facilita el mantenimiento y soporte del proyecto.
- Proporciona amplia reutilización. Al separar de manera clara la lógica de negocio (capa lógica) de la vista (capa presentación) permite la reusabilidad de la primera, de modo que la misma implementación de la lógica de negocio que maneja una aplicación pueda ser usado en otras aplicaciones, sean éstas Web o no.
- Mayor flexibilidad (se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad).
- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.

Desventajas:

- Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de performance pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.
- Además, los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples.

2.6 Conclusiones

En este capítulo se define la propuesta que le da solución al problema científico, en este caso se define la utilización del Patrón 3 Capas como arquitectura para la producción de software educativo en la UCI.

CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

3.1 Introducción

En el presente capitulo se procederá a validar la propuesta obtenida en este trabajo de investigación, la variante que se utilizará es el método Delphy. Se espera según los resultados que arroje el método, la validación de la propuesta.

3.2 Método para la validación de la propuesta.

Con el objetivo de validar la propuesta obtenida en este trabajo de investigación, se utilizó la variante del método Delphy propuesta por Silvia Colunga y Georgina Amayuela.

La hipótesis es una conjetura sobre la posible solución de un problema; esto es, un pronóstico. La pronosticación de un hecho o fenómeno puede clasificarse según su posibilidad de ocurrencia en:

- Pronóstico de previsión: los elementos del fenómeno futuro son, en su mayor parte, conocidos.
- Pronóstico de predicción: los elementos del fenómeno futuro son generalmente desconocidos.

Características del pronóstico:

- Debe reflejar el tiempo o intervalo de tiempo en que ha de tener lugar la situación pronosticada.
- Valoración del grado de ocurrencia del fenómeno.
- Posibilidad de comprobación.

El pronóstico se apoya en dos tipos fundamentales de métodos.

- Los de base objetiva.
- Los de base subjetiva.

Estos últimos se conocen con el nombre de criterios de expertos.

Este método se caracteriza por:

- Anonimato.
- Respuesta estadística del grupo.
- Retroalimentación controlada.

SECUENCIA A TENER PRESENTE EN EL CRITERIO DE EXPERTOS

Selección de los expertos.

a) Coeficiente de competencia. Este coeficiente se determina mediante la fórmula: K= ½ (kc + ka), donde kc es el coeficiente de conocimientos y ka es el coeficiente de argumentación.

El presunto experto marcará en la casilla enumerada, según su criterio acerca de la capacidad que él tiene sobre el tema que se la ha sometido a su consideración, en una escala del 0 al 10 y que después para ajustarla a la teoría de las probabilidades se multiplicará por 0,1; de esta forma, la evaluación "0" indica que el experto no tiene absolutamente ningún conocimiento de la problemática correspondiente, mientras que la evaluación "10" significa que el experto tiene pleno conocimiento de la problemática tratada.

b) Para calcular el coeficiente de argumentación se procede de la siguiente forma:

El experto debe marcar, según su criterio, su grado de competencia sobre los aspectos (fuentes de argumentación) sometidos a consideración.

Las marcas de los expertos se traducen a puntos. Ver anexo 5.

Con estos elementos (Kc y Ka) es suficiente para obtener el coeficiente de competencia K, según la fórmula vista anteriormente. La forma descrita con anterioridad nos permite seleccionar la competencia de nuestros expertos.

- Si 0.8 < k <1.0, el coeficiente de competencia es alto.
- Si 0.5 < k < 0.8, el coeficiente de competencia es medio.
- Si k < 0.5 el coeficiente de competencia es bajo.

Una vez obtenido el coeficiente de competencia se decidirá que expertos deben ser seleccionados, preferentemente si este es alto o medio para obtener resultados positivos. Para este trabajo se utilizaron 8 expertos: 4 con coeficiente de competencia medio y 4 con coeficiente de competencia alto. Ver las tablas que recoge la autovaloración del posible experto. Anexo 6

Logrado ya el número de expertos, se buscan sus criterios sobre la temática sometida a consideración. Las preguntas a formular no deben ser demasiadas, pero si sobre cuestiones medulares sobre la investigación que se realiza. Se tuvieron en cuenta cinco indicadores que forman parte de la propuesta para una arquitectura en el desarrollo del software educativo en la UCI. Además se pidió que evaluaran

los pasos en las categorías de: muy adecuada (MA), bastante adecuada (BA), adecuada (A), poco adecuada (PA) y no adecuada (NA). Ver anexo 7

Los resultados se recogen en una tabla de doble entrada (ver anexo 8), teniendo en cuenta la cantidad de veces que un indicador es señalado con una categoría determinada, en las encuestas realizadas a los expertos.

Tabulados los datos, se realizan los siguientes pasos para obtener los resultados deseados:

- Primer paso: Se construye una tabla (ver anexo 9) de frecuencias acumuladas. Esto es, cada número en la fila, excepto el primero se obtiene sumándole el anterior.
- Segundo paso: Se copia la tabla anterior y se borran los resultados numéricos. Ahora, en esta nueva tabla, se construye la tabla de frecuencias relativas acumulativas. Marque el cuadro donde va a poner el resultado. Esta tabla se logra dividiendo por n (número total de expertos) cada uno de los números de la tabla anterior.
- Tercer paso: Buscar las imágenes de los elementos de la tabla anterior por medio de la función (Dist. Normal. Standard Inv) que trae el Excel. Para lograr este objetivo, copiar la tabla anterior y eliminar los números a fin de colocar los nuevos elementos.
- Cuarto paso: A la misma tabla se adiciona tres columnas y una fila para colocar los resultados de la suma de las columnas, suma de filas, promedio de las columnas, promedios de las filas.

Para hallar N, se divide la suma de las sumas entre X. Esta X se ha obtenido de multiplicar el número de categorías por el número de preguntas. Al dividir la suma de las suma entre X, se obtiene un resultado tomado con una sola cifra decimal.

El valor N-P nos da el valor promedio que otorgan los expertos consultados para cada pregunta de la propuesta.

Las sumas obtenidas en las cuatro primeras columnas nos dan los puntos de cortes. Los puntos de corte nos sirven para determinar la categoría o grado de adecuación de cada paso de la propuesta según la opinión de los expertos consultados. Ver anexo 10

A partir de las cifras arrojadas con el método Delphy, se obtuvieron los resultados siguientes sobre los indicadores de la propuesta:

1. Objetivos que persigue la propuesta. MA
2. Representación de los elementos arquitectónicos más significativos en la capa de presentación. BA
3. Representación de los elementos arquitectónicos más significativos en la capa de lógica. BA

4. Representación de los elementos arquitectónicos más significativos en la capa de acceso a datos. BA
5. Aplicabilidad de la propuesta. BA

Se puede comprobar que los resultados obtenidos son satisfactorios según el criterio de los expertos consultados, al no tener ningún indicador poco adecuado o no adecuado. Con esto se pueden dar por concluidos en cuanto a su elaboración teórica.

3.3 Conclusiones

En este capitulo se utilizó el método Delphy o criterio experto para validar la propuesta de la investigación, los resultados obtenidos fueron satisfactorios.

CONCLUSIONES GENERALES

El desarrollo de la arquitectura de software es una de las etapas fundamentales en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema.

Es de vital importancia que todo sistema de software esté respaldado por una arquitectura sólida que facilite el entendimiento del mismo, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.

Se realizó un estudio profundo acerca de varios estilos y patrones arquitectónicos que existen en la actualidad, para hacer uso de las mejores técnicas de diseño arquitectónico.

Después de un análisis de los elementos estructurales que se utilizan en la producción de software educativo en la UCI, se propuso la utilización del Patrón en 3 Capas como arquitectura para el desarrollo de los mismos.

Se cumplieron todos los objetivos trazados en esta investigación y a la vez que se dio respuesta a la pregunta planteada en el problema científico a partir de definir los elementos arquitectónicos mas significativos en la producción de software educativo en la UCI.

RECOMENDACIONES

Identificar una estructura única de arquitectura para software y multimedias educativas suele ser compleja debido a la gran variedad de estos, que se producen actualmente en la Universidad. Se hace necesario realizar el siguiente conjunto de recomendaciones para encaminar de esta forma investigaciones futuras en el propio ámbito de la actual y mejorar los resultados científicos que se han obtenido:

1. Enriquecer los elementos conceptuales de la nueva propuesta de arquitectura, sobre la base de conceptos de la Programación Orientada a Objetos, para brindarle a dicha arquitectura una mayor solidez en el tratamiento de la información.
2. Aplicar la nueva propuesta de arquitectura a los proyectos productivos de software educativo en la UCI.
3. Conformar con especialistas de la producción de software educativo en la UCI, investigadores del área del conocimiento y desarrolladores de otras instituciones a fines, un grupo de estudio y desarrollo de la propuesta para lograr obtener nuevas versiones superiores y escalables rápidamente y que se adapten a los requerimientos de las multimedias educativas.

REFERENCIAS BIBLIOGRÁFICAS

1. Shaw, M., y D. Garlan. Software Arquitecture. s.l. : Prentice Hall, 1996.

2. Dijkstra, Edsger. “The Structure of the Multiprogramming system.” Communications of the ACM, 26(1), pp.49-52. Enero de 1983.

3. Clements Paul “A Survey of Architecture Description Languages”. Alemania, 1996.

4. Díaz-Antón, María Gabriela, y otros. [Online] 2002. [Cited: octubre 14 , 2005.] http://www.infedu.coord.usb.ve/proyectos/proyecto3.html.

5. Marqués, P. "La informática como medio didáctico: software educativo, posibilidades e integración curricular.". 1999. Vols. 93-109 DOI:.

6. Fernández, A. El formador de Formación Profesional y Ocupacional. Barcelona : Octaedro, 2000.

7. Autores, Colectivo de. Software Educativo. [Online] 1998. [Cited: abril 26, 2006.] http://cavalletto.freeservers.com/pag%20educativa%20de%20amortiza.htm.

8. Kruchten, Philippe. “The 4+1 View Model of Architecture”. 1995. pp. 42-50.

9. Grady Booch, James Rumbaugh e Ivar Jacobson. El Lenguaje Unificado de Modelado. Madrid : Addison-Wesley, 1999.

10. Shaw, Mary. “Some Patterns for Software Architecture,” en Pattern Languages of Program Design. Reading : Addison-Wesley, 1996. pp. 255-269. Vols. 2, J. Vlissides, J. Coplien, y N. Kerth (eds.).

11. Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. “Stylized architecture, design patterns, and objects”.

12. Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. “Architectural Styles, design patterns, and objects”. s.l. : IEEE Software, 1997. pp. 43-52.

13. Lemus-Olalde, Sharon White y Cuauhtémoc. “The software architecture process”. [Online] 1997. http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf.

14. Pressman, Roger S. "Ingeniería de Software. Un enfoque práctico." 5ta Edición (traducción de la edición original en inglés "Software Engineering. A practical approach"). Madrid : Mac Graw Hill, 2001.

15. Grady Booch, James Rumbaugh e Ivar Jacobson. El Lenguaje Unificado de Modelado. Madrid : Addison-Wesley, 1999.

16. Dewayne E. Perry, A. L. W. "Foundations for the study of software architecture". 1992.

17. Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. “Architectural Styles, design patterns, and objects”. s.l. : IEEE Software, 1997. pp. 43-52.

18. G, Booch. Software Architecture and the UML. [Online] 1998. http://www.rational.com/uml como arch.zip.

19. Lemus-Olalde, Sharon White y Cuauhtémoc. “The software architecture process”. [Online] 1997. http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf.

20. MARTIN, R. C. Design Principles and Design Patterns.

21. Microsoft Patterns & Practices. 2004.

22. Stefan, Sauer and Engels, Gregor. “Extending UML for Modeling of Multimedia Applications”. [Online] 2004. http://wwwcs.upb.de/cs/agengels/Papers/2003/EngelsSauerNeu-HCC03.pdf.

23. Larman, Craig. “UML y Patrones. Introducción al análisis y diseño orientado a objetos”. 2da Edición. México : Prentice Hall Hispanoamericana, 1999.

24. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. “A system of Patterns”. West Sussex, Inglaterra : Wiley, 1996.

25. Seshadri, G. ”Understanding Java Server Pages Model 2 Architecture. Exploring the MVC design pattern”. [Online] 1999. http://www.javaworld.com/javaworld/jw-12-1999/.

26. Aurelia, B. G. La Responsabilidad del Arquitecto. 2004.

27. Valverde Berrocoso, J. "Diseño y elaboración de un programa educativo multimedia". s.l. : Universitas Editorial, 2000. p. 273 y ss.

28. Barrera Yanes, Rafael. Del objetivo al guión interactivo. [ed.] GIGA. La Habana : s.n., 1998. Vols. 1, 1998.

BIBLIOGRAFÍA

Aurelia**,** B. G. *La Responsabilidad del Arquitecto.* 2004.

Autores, Colectivo de. Software Educativo. [Online] 1998. [Cited: abril 26, 2006.] http://cavalletto.freeservers.com/pag%20educativa%20de%20amortiza.htm.

Barrera Yanes, Rafael. *Del objetivo al guión interactivo.* [ed.] GIGA. La Habana : s.n., 1998. Vols. 1, 1998.

Bass, L., P. Clements y R. Kazman. *Software Architecture in Practice.* s.l. : Addison-Wesley, 1998.

Buschmann, F. *Pattern-Oriented Software Architecture.* s.l. : John Wiley & Sons ISBN: 0471958697, 1996.

Dewayne E. Perry, A. L. W. *"Foundations for the study of software architecture".* 1992.

Díaz-Antón, María Gabriela, y otros. [Online] 2002. [Cited: octubre 14 , 2005.] http://www.infedu.coord.usb.ve/proyectos/proyecto3.html.

Dijkstra, Edsger. *"The Structure of the Multiprogramming system." Communications of the ACM, 26(1), pp.49-52.* Enero de 1983.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *"A system of Patterns".* West Sussex, Inglaterra : Wiley, 1996.

Fernández, A. *El formador de Formación Profesional y Ocupacional.* Barcelona : Octaedro, 2000.

http://wwwcs.upb.de/cs/agengels/Papers/2003/EngelsSauerNeu-HCC03.pdf.

G, Booch. Software Architecture and the UML. [Online] 1998.

http://www.rational.com/uml como arch.zip.

Grady Booch, James Rumbaugh e Ivar Jacobson. *El Lenguaje Unificado de Modelado.* Madrid : Addison-Wesley, 1999.

IEEE-1471 (2000) Recommended Practice for. Architectural Description of Software-Intensive Systems.

Kruchten, Philippe. *"The 4+1 View Model of Architecture".* 1995. pp. 42-50.

Larman, Craig. *"UML y Patrones. Introducción al análisis y diseño orientado a objetos".* 2da Edición. México : Prentice Hall Hispanoamericana, 1999.

Lemus-Olalde, Sharon White y Cuauhtémoc. "The software architecture process". [Online] 1997. http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf.

Marqués, P. *"La informática como medio didáctico: software educativo, posibilidades e integración curricular.".* 1999. Vols. 93-109 DOI:.

MARTIN, R. C. *Design Principles and Design Patterns.*

Microsoft Patterns & Practices. 2004.

Pressman, Roger S. *"Ingeniería de Software. Un enfoque práctico." 5ta Edición (traducción de la edición original en inglés "Software Engineering. A practical approach").* Madrid : Mac Graw Hill, 2001.

Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. *"Stylized architecture, design patterns, and objects".*

Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. *"Architectural Styles, design patterns, and objects".* s.l. : IEEE Software, 1997. pp. 43-52.

Seshadri, G. "Understanding Java Server Pages Model 2 Architecture. Exploring the MVC design pattern". [Online] 1999.
http://www.javaworld.com/javaworld/jw-12-1999/.

Shaw, M., y D. Garlan. *Software Arquitecture.* s.l. : Prentice Hall, 1996.

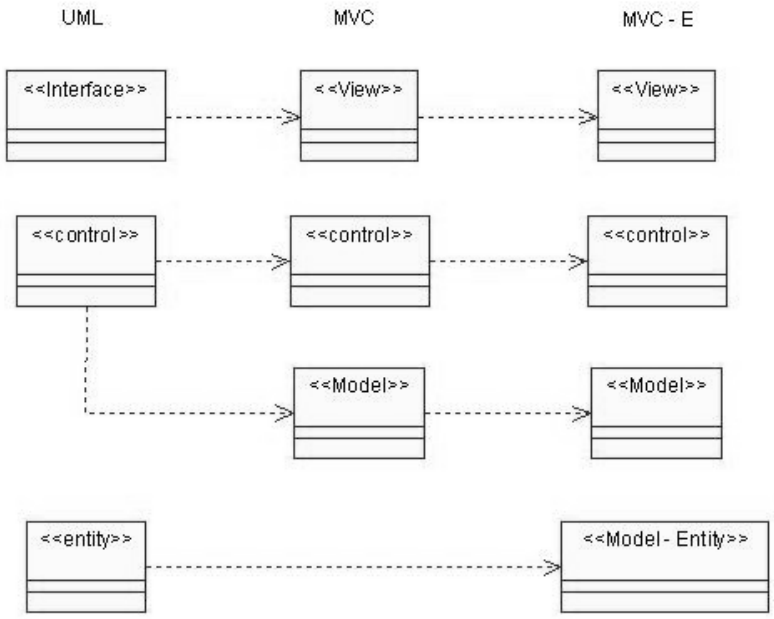
Shaw, Mary. *"Some Patterns for Software Architecture," en Pattern Languages of Program Design.* Reading : Addison-Wesley, 1996. pp. 255-269. Vols. 2, J. Vlissides, J. Coplien, y N. Kerth (eds.).

Silvia Colunga, G. A. La Psicología Educativa, su objeto, métodos y problemas principales. , Universidad de Camagüey, 2003. p.

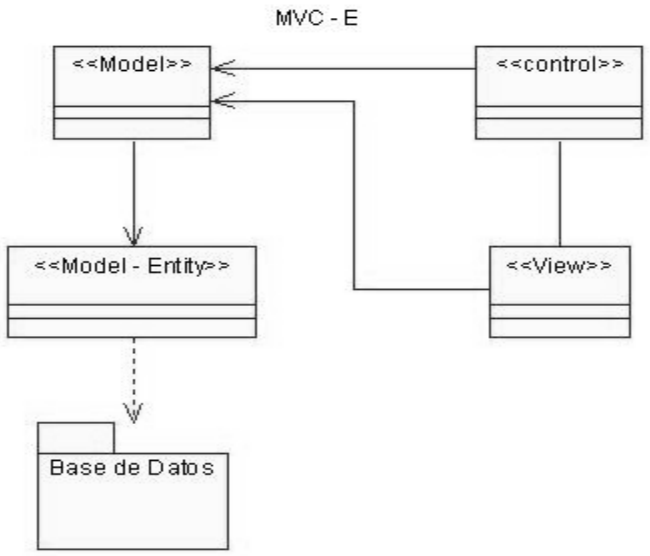
Stefan, Sauer and Engels, Gregor. "Extending UML for Modeling of Multimedia Applications". [Online] 2004.

ANEXOS

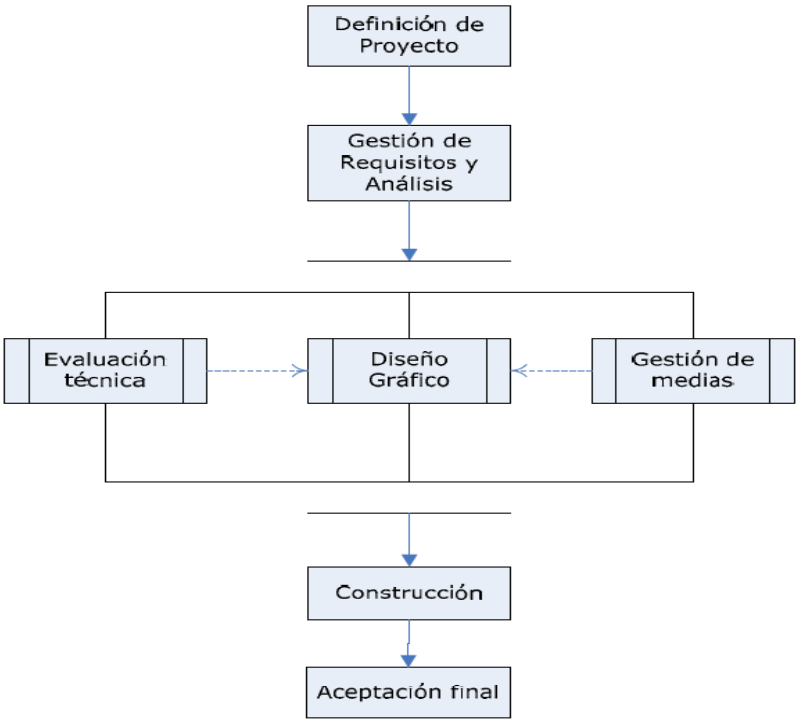
Anexo 1: Correspondencia entre UML – MVC y MVC – E.



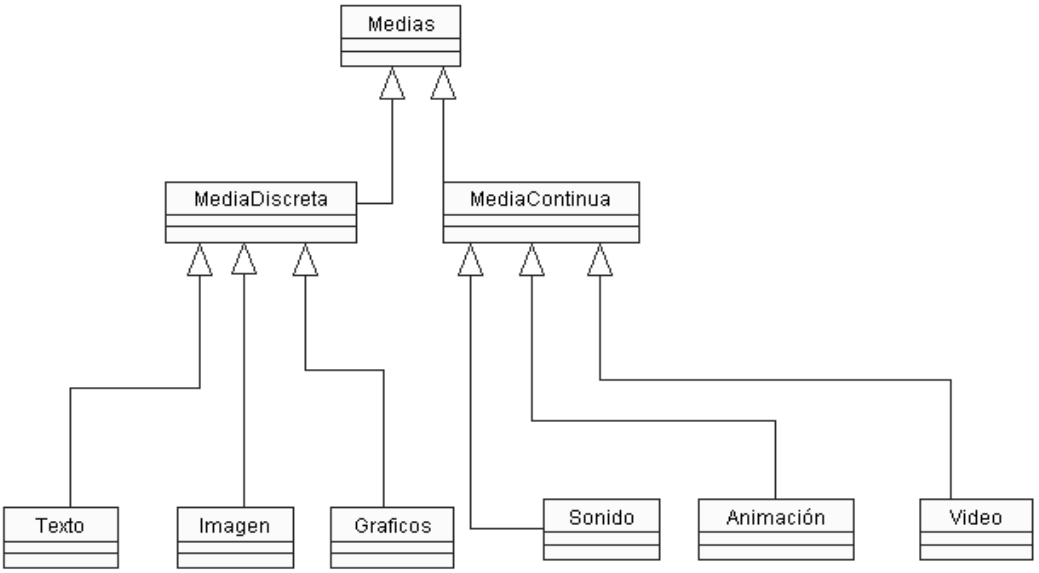
Anexo 2: Esquema del MVC – E.



Anexo 3: Flujo General de Trabajo de la línea de Software Educativo



Anexo 4: Jerarquía de Medias.



Anexo 5

Tabla 1 - Escala de grado de competencia.

FUENTES DE ARGUMENTACION	Grado de Influencia de cada una de las fuentes en sus criterios.		
	A (Alto)	M (Medio)	B (Bajo)
Análisis teóricos realizados por usted.	0.3	0.2	0.1
Su experiencia obtenida.	0.5	0.4	0.2
Trabajo de autores nacionales.	0.05	0.05	0.05
Trabajo de autores extranjeros.	0.05	0.05	0.05
Su propio conocimiento del estado del problema en el extranjero.	0.05	0.05	0.05
Su intuición.	0.05	0.05	0.05
Totales	1.0	0.8	0.5

Anexo 6

ENCUESTA DE AUTOVALORACIÓN

Compañero (a):

En la ejecución de la presente tesis, deseamos someter a la valoración de un grupo de expertos, proponer los elementos arquitectónicos que se utilizan en la producción de software educativo en la UCI. Para ello necesitamos conocer el grado de dominio que Ud. posee del análisis y diseño del software educativo en la UCI; y con ese fin deseamos que responda lo que se le pide a continuación.

Nombre y apellidos: _____

Centro de trabajo: _____

Labor que realiza: _____

Años de experiencia: _____Especialidad: _____

Categoría docente: _____Categoría científica: _____

País: _____

1.- Marque con una cruz (X) el grado de conocimiento que Ud. tiene sobre el proceso que se investiga:

0	1	2	3	4	5	6	7	8	9	10

2.- Marque con una cruz (X) las fuentes que le han servido para argumentar el conocimiento que tiene Ud. de la temática que se investiga. Encierre en un círculo la que más ha influido.

No.	Fuentes de argumentación	Grado de influencia		
		Alto	Medio	Bajo
1.-	Análisis realizado por Ud.			
2.-	Experiencia.			
3.-	Trabajos de autores nacionales.			
4.-	Trabajos de autores extranjeros.			
5.-	Su propio conocimiento del tema.			
6.-	Su intuición.			

Anexo 7

ENCUESTA A EXPERTOS.

Compañero (a):

La presente tesis tiene por objetivo proponer los elementos arquitectónicos (una arquitectura) a utilizar en la producción de software educativo en la UCI.

Necesitamos que lea detenidamente el documento anexo y valore el grado de **factibilidad** de los elementos que conforman la propuesta. Coloque una X según el nivel que usted cree que posee.

Tabla 2 – Encuesta a expertos.

No	Indicadores para medir rasgos de carácter que determinan comportamientos ante situaciones	MA	BA	A	PA	NA
1.	Objetivos que persigue la propuesta.					
2.	Representación de los elementos arquitectónicos más significativos en la capa de presentación.					
3.	Representación de los elementos arquitectónicos más significativos en la capa de lógica.					
4.	Representación de los elementos arquitectónicos más significativos en la capa de acceso a datos.					
5.	Aplicabilidad de la propuesta.					

3.1.- Determine si los elementos anteriores son:
Necesarios si__ no___ no sé___
Suficientes si__ no___ no sé___
a) Si lo considera conveniente, proponga otros:

3.2.- Exprese otros criterios o recomendaciones que pudieran servir para perfeccionar los indicadores propuestos.

Anexo 8

Tabla 3 - Tabla de doble entrada.

Indicadores	C1 Muy Adecuado	C2 Bastante Adecuado	C3 Adecuado	C4 Poco Adecuado	C5 No Adecuado	Total
I1						
I2						
…						
In						
Total de aspectos a validar						

Anexo 9

Tabla 4 - Tabla de frecuencias acumuladas

Indicadores	C1 Muy Adecuado	C2 Bastante Adecuado	C3 Adecuado	C4 Poco Adecuado	C5 No Adecuado
I1					
I2					
…					
In					

Anexo 10

Tabla 5 - Tabla con los puntos de corte.

Indicadores	C1 Muy Adecuado	C2 Bastante Adecuado	C3 Adecuado	C4 Poco Adecuado	Suma	P	NP
I1							
I2							
In							
Suma							
Punto de Corte							

Anexo 11

Entrevista realizada a los líderes de los proyectos Libros Electrónicos, Historia Universal y MENPET, dedicados al desarrollo del software educativo en la UCI.

- ¿Cuáles fueron las herramientas verticales y horizontales que se utilizaron en el producto?
- ¿Cuántos módulos componen el producto y cuáles son?
- ¿Cuántos paquetes contiene el producto y que contiene cada paquete?
- ¿Se utiliza algún tipo de arquitectura para el desarrollo del producto?
- ¿Qué artefactos se generan o utilizan en el producto?
- ¿Qué artefactos o elementos de estos deben de mantenerse en futuras soluciones?

GLOSARIO DE TÉRMINOS

IBM: (**I**nternational **B**usiness **M**achines) es una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

Software: (soporte lógico) los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.

Informática: es la disciplina que estudia el tratamiento automático de la información utilizando dispositivos electrónicos y sistemas computacionales. Es la unión sinérgica del cómputo y las comunicaciones.

Servicios: es un conjunto de actividades que buscan responder a una o más necesidades de un cliente.

Servidor: Una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

Cliente/Servidor: Esta arquitectura consiste básicamente en que un programa (cliente informático) realiza peticiones a otro programa (servidor) que les da respuesta.

Servicio WEB: (en inglés *Web services*) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores.

IEEE: corresponde a las siglas de The **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación e ingenieros en telecomunicación.

XML: (sigla en inglés de e**X**tensible **M**arkup **L**anguage) Lenguaje de marcado extensible es un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Recursos: Conjunto de elementos disponibles para resolver una necesidad o llevar a cabo una tarea.

MVC: el Modelo Vista Controlador es un patrón de arquitectura que divide una aplicación interactiva en 3 áreas: procesamiento, salida y entrada.

Medias: se refiere a recursos audiovisuales, por ejemplo audio, sonidos y video.

Experto: Es el individuo o grupo de individuos, organizaciones, etc., capaces de ofrecer valoraciones conclusivas de un fenómeno determinado y hacer recomendaciones respecto a sus momentos fundamentales con un máximo de competencia. (SILVIA COLUNGA 2003)

Útiles: utilerías indispensables que debe contener cualquier herramienta multimedia para desarrollar algunas tareas, peculiares pero repetidas con frecuencia. Estos son los accesorios conformables y bien empleados que hacen más fácil su vida con la PC.

POSA: Pattern Oriented Software Architecture.

SEI: Software Engineering Institute.

PC: (en inglés Personal Computer) Computadora Personal.

TIC: Tecnologías de la Información y las Comunicaciones.