Universidad de las Ciencias Informáticas

"FACULTAD 8"

Arquitectura de la Herramienta para la Modelación de Aplicaciones Educativas: ApEM-H 1.0

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Davel Jesús Rodríguez Vento.
Oriel Alian Sorzano De La Torre.

Tutor: Ing. Damir Góngora Mora.

Co-Tutor: M. Sc. Febe Ciudad Ricardo.

Ciudad de la Habana, Junio de 2008.

"Año 50 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 8 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Davel Jesús Rodriguez Vento

_____

Firma del autor.

Oriel Alian Sorzano de La Torre

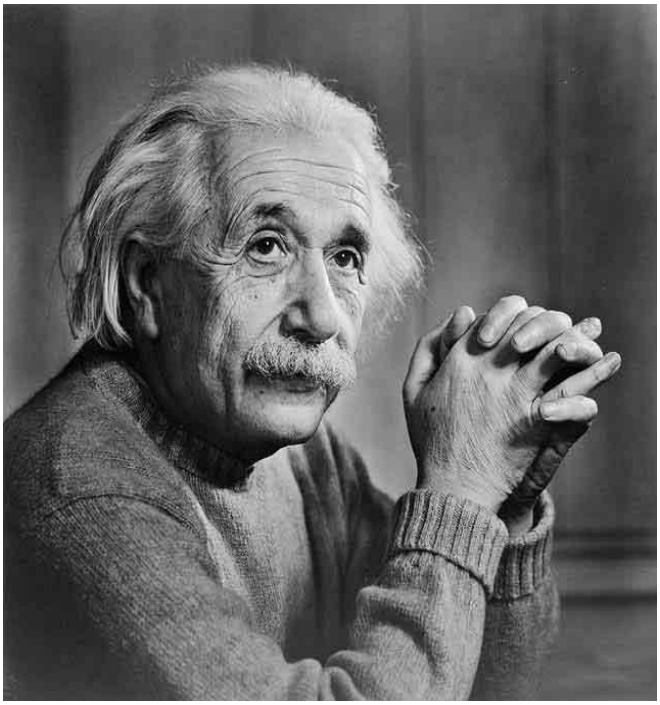
_____

Firma del autor.

Ing. Damir Góngora Mora.

_____

Firma del tutor.



"Intenta no volverte un hombre de éxito, sino volverte un hombre de valor."

-Albert Einstein.

AGRADECIMIENTOS

Davel

A toda mi familia: En especial a mis padres, a mi hermano David, a mis abuelos Olga ("mima") y Gil ("papa"), a mis tías Odalis, Olgui, Yely, a mis tíos, a mis primos. A todos muchas gracias por su apoyo constante e incansable.

A mis amigos: A los "viejos" amigos (en especial a "Lulú", Melvin, "Pedry", Mario, Daisbel, Deivis, "Roly") por estar siempre ahí, a pesar del tiempo. A los "nuevos" amigos (En especial a David, por ayudarme con el documento y a Oriel mi compañero de tesis, por poner el máximo en este trabajo). Por los buenos momentos, por ayudarme durante estos "cortos" cinco años siempre que lo necesité, por la compañía, por ser ahora parte de mi vida, y también por las "convo" que no podían faltar, a todos ustedes doy gracias.

A María del Carmen y "Mayito", por sus consejos en lo personal y en lo docente, a ambos gracias por su bondad interminable.

A Claudia. Por existir, por su ternura, por mostrarme el amor verdadero, por haberme salvado.

Oriel

A mi mamá y papá por la educación que me dieron y todo lo que me enseñaron.

A mi hermano Osiel por ser siempre ejemplo para mí.

A mis tíos, en especial a mi tía Angelina, que ha dedicado su vida en estos 5 años para mí.

En general a toda mi familia y amigos.

Al compañero de tesis Davel que se esforzó machísimo en esta Investigación y al tutor Damir por su apoyo.

DEDICATORIA

Davel

A todos los que de alguna manera han estado detrás de tanto trabajo durante tanto tiempo. A los que me han ayudado durante la carrera, tanto en el plano docente como personal. A ellos me debo y gracias a ellos he podido llegar hasta aquí.

Oriel

A mi mama y papa por la educación que me dieron y todo lo que me enseñaron.

A mi hermano Osiel por ser siempre ejemplo para mí.

A mis tíos, en especial a mi tía Angelina, que ha dedicado su vida en estos 5 años para mí.

En general a toda mi familia y amigos.

Al equipo de béisbol de Santiago de Cuba.

RESUMEN

En el presente trabajo se propone un modelo arquitectónico a seguir durante el desarrollo de la herramienta CASE ApEM-H que se encargará de generar toda la documentación ingenieril modelada por el lenguaje ApEM-L. De aquí que el objetivo fundamental sea la definición de la misma. Para conseguir tal objetivo se ha centrado la investigación en el estudio de las herramientas CASE, la arquitectura en la cual se centran dichas herramientas y la arquitectura de software en general. Esta investigación se hizo necesaria debido a que el tema que trata, la arquitectura de software, es considerada la columna vertebral de cualquier proyecto de software que se quiera desarrollar, de ahí la importancia que se le concede.

TABLA DE CONTENIDO

INTRODUCCIÓN

En las últimas tres décadas se ha desarrollado un vertiginoso desarrollo de la incorporación de la informática al entorno educativo, dando surgimiento al área de la Informática Educativa y con ella a conceptos como Software Educativo, que se hace muy común en nuestros días. El propio desarrollo de este tipo de software ha traído como consecuencia variaciones importantes en los flujos de los procesos productivos y en los artefactos de modelación para generar su documentación en el uso de las tecnologías multimedia e hipermedia.

El ámbito informático actual para el desarrollo de software educativo, establece grandes retos a las metodologías y procesos de desarrollo y lenguajes notacionales de software existentes (RUP, UML), ambos con una aceptación generalizada; entre los que se encuentra la representación de las características pedagógicas y funcionales de este tipo de aplicaciones, máxime con las características y los pilares de la pedagogía cubana [1].

En la última década, se extendió el lenguaje notacional UML a OMMMA-L (Lenguaje Orientado a Objetos para la Modelación de Aplicaciones Multimedia) para el tratamiento de aplicaciones educativas multimedia, facilitando el modelado de un gran rango de aspectos de aplicaciones multimedia interactivas de una forma integrada y comprensiva, respetando igualmente los estándares establecidos ya por el primer lenguaje mencionado. No obstante, por las propias condiciones descritas anteriormente, uno de los grandes retos para el establecimiento y utilización definitiva de este lenguaje es la no posibilidad de representación de la totalidad de las características del software educativo cubano y sus entornos de desarrollo. La primera versión del Lenguaje Unificado de Modelado (UML) para la modelación de software de propósito general orientado a objetos, fue de un gran impacto en la industria internacional y tuvo una excelente aceptación, al punto de convertirse en un estándar mundial en este sentido [1].

Dado que “Desafortunadamente, UML no soporta todos los aspectos de las aplicaciones multimedia de una manera adecuada e intuitiva. Especialmente, características del lenguaje para el modelado de aspectos de la interfaz de usuario no están explícitamente proporcionadas. Otros conceptos de UML, no son lo suficientemente maduros o son muy poco gráficos y esto agrava la modelación de multimedia innecesariamente.” (Sauer, y otros, 2001) y que en Cuba ya se han acumulado más de 20 años de experiencia en la elaboración de software educativo, en el transcurso de esos 20 años de experiencia acumulados en el desarrollo de software educativo en nuestro país, el resto de la producción de software y en menos medida el desarrollo de estos software educativos, ha

transitado por diferentes paradigmas en la programación como son: el estructurado, el orientado a eventos y el orientado a objeto, las principales preocupaciones en este tiempo no han estado en las características informáticas (dicho de algún modo) de este tipo de aplicaciones, sino en sus posibilidades educativas y pedagógicas, es precisamente la UCI el "(…) punto de enlace entre el Sistema de Centros de Educación Superior y de la Ciencia con la Industria propiciando la transferencia tecnológica" (MIC, 2003), es donde mejores condiciones creadas existen para unificar los esfuerzos y conocimientos existentes en el desarrollo de software educativos, impulsando la inserción continua de mejoras en el proceso de creación de este tipo de aplicaciones y en especial la modelación de las mismas; así como sus análisis y sus diseños.

Por tales motivos es que surge la concepción del Lenguaje para la Modelación de Aplicaciones Educativas "ApEM – L" el cual tiene como principales objetivos:

Desarrollar una extensión del lenguaje de modelado UML, tomándolo como base e incorporando a este, a través de sus mecanismos de extensión, los elementos fundamentales del proceso productivo UCI. De esta forma se produjo un lenguaje de propósito particular para la modelación de aplicaciones educativas.

Incorporar los elementos más significativos de extensiones anteriores como OMMMA-L (2001) y a su vez respetar lo establecido por el estándar OCL (2003), para de esta forma lograr una extensión consistente y escalable en el tiempo [1].

No complicar ApEM – L con elementos que lo convirtieran o abarcaran un método de desarrollo de aplicaciones educativas, sino solo el área de la representación y la documentación de este tipo de aplicaciones.

No circunscribir ApEM – L a un proceso de desarrollo en específico, sino expresarlo de manera tal que pueda ser utilizado con cualquiera de los existentes, aunque se sugiere la utilización de procesos de desarrollo, iterativos, incrementales y basados en prototipos, que permitan la modelación de sistemas orientados a objetos [1].

Como **situación problémica** se tiene que el lenguaje de modelación de aplicaciones educativas ApEM-L provee una notación para representar los elementos estructurales, lógicos, funcionales, pedagógicos y de patrones de ingeniería al modelar el software educativo en la UCI. Sin embargo la herramienta CASE (ApEM-H) concebida para modelar este lenguaje, y así permitir una mejor documentación técnica y una más clara representación ingenieril en este tipo de aplicaciones, no cuenta con una arquitectura definida que pueda resumir los elementos significativos y generalizables, desde el punto de

vista arquitectónico existentes hoy en el entorno productivo de la universidad, lo cual hace necesario una investigación lo suficientemente exhaustiva que pueda, a través de el estudio de las diferentes herramientas CASE existentes, llegar a la definición de la misma.

En estas condiciones se identificó como **problema científico** escasez de documentación e información de la arquitectura de las herramientas informáticas para el diseño asistido por computadora (CASE), lo que dificulta la definición correcta de un modelo arquitectónico e ingenieril para ApEM-H.

El problema descrito genera como **objeto de estudio** la arquitectura de las herramientas informáticas para el diseño asistido por computadora (CASE), enmarcando los resultados científicos de la investigación en el área de los proyectos de desarrollo y producción de software educativos de la UCI como **campo de acción**.

Para darle solución a este problema se plantea como **objetivo general** la definición de la arquitectura de desarrollo de la Herramienta para la Modelación de Aplicaciones Educativas: ApEM - H, en el contexto productivo cubano.

Para facilitar el cumplimiento de este objetivo se ha decidido desglosarlo en los siguientes **objetivos específicos**:

1. Identificación de los elementos significativos y generalizables, desde el punto de vista arquitectónico de las herramientas (CASE), existentes hoy en el entorno productivo cubano.
2. Determinación de los elementos establecidos por los estándares que debe incorporar ApEM - H.

Se plantea como **hipótesis** del presente trabajo que si se identifican los elementos significativos y generalizables, desde el punto de vista arquitectónico de las herramientas (CASE), así como los establecidos por los estándares que debe incorporar ApEM-H, entonces quedará definida la arquitectura con la cual se creará un esbozo en el cual se centrará el desarrollo de la herramienta.

La investigación científica se guió por el conjunto de **tareas** que se describen a continuación para darle cumplimiento a los objetivos planteados:

1. Estudiar la documentación existente sobre el tema en Cuba y el mundo.

2. Estudiar las exigencias establecidas por los estándares nacionales e internacionales que se aplican al producto.

3. Estudiar las características arquitectónicas de las herramientas CASE utilizadas en el entorno productivo cubano.

4. Identificar los elementos arquitectónicos significativos de las herramientas CASE utilizadas en el entorno productivo cubano.

5. Estudiar para su incorporación en la arquitectura de ApEM-H, lo establecido por los patrones MVC, MVC_{mm} y MVC-E.

6. Definir la arquitectura que tendrán la herramienta para la Modelación de Aplicaciones Educativas en Cuba.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En el presente capítulo se explicarán los métodos y estrategias científicos utilizados durante la investigación así como la fundamentación de los mismos. De igual manera se ha plasmado todo lo concerniente a las Herramientas CASE, dígase características de las mismas, rango que abarcan dentro del ciclo de vida del desarrollo del software, clasificaciones más comunes. Dentro de este tipo de herramientas se ha hecho énfasis en las CASE-Integradas, destacando de estas, cuales son las más utilizadas en el ámbito internacional y sus principales características.

1.2 Estrategia y métodos científicos utilizados en la investigación.

La estrategia utilizada en esta investigación es la **Exploratoria** ya que se hace a partir de una situación problemica dada y no se tiene una idea clara del asunto en cuestión. Mediante la misma el principal objetivo ha sido familiarizarse con la temática objeto de estudio, la situación en que se encuentra y los métodos y técnicas utilizados en su ejecución.

Para realizar esta investigación se ha comenzado por buscar las fuentes de información que permitan conocer la situación actual de la temática en estudio y definir su factibilidad de ejecución, elaborar el diseño de investigación más apropiado y seleccionar o elaborar las técnicas necesarias para la obtención de los datos.

La fuente de información más apropiada es la acumulada en la bibliografía existente y la práctica que se obtiene de los sujetos que están vinculados a la problemática que se ha investigado.

Para seleccionar las fuentes bibliográficas más apropiadas se ha tenido en cuenta el nivel de influencia que pueden tener en el tema que se investiga la calidad y actualidad de sus contenidos y el prestigio y autoridad de los autores estudiados. Por ejemplo se han citado

fragmentos de expertos en el tema como Pressman, lo cual da cierto nivel de cientificidad a la investigación.

Se ha organizado la lectura de la información de acuerdo con las necesidades de los investigadores, con un orden lógico que ha permitido lograr un dominio considerable del campo científico en que se investiga.

Para registrar la investigación obtenida se han utilizado técnicas como son las fichas bibliográficas y de contenido, lo que ha facilitado su ordenamiento y análisis posterior para elaborar el fundamento que aportara la base teórica y metodológica necesaria para realizar la investigación científica.

La otra fuente de información en la investigación ha sido la obtención de datos prácticos iniciales acerca del objeto de estudio a través de sujetos y situaciones propia de la problemática estudiada y de personas que por su experiencia pueden brindar información real sobre el asunto investigado.

La combinación de estas dos fuentes de información, ha permitido establecer si la problemática planteada ya ha sido investigada en el campo de acción de los investigadores, o si la teoría existente es suficiente para dar una respuesta sin necesidad de hacer la investigación, y al concluir que ninguno de estos casos se ha verificado, se ha llevado a cabo la investigación.

El método de investigación que se ha aplicado es el método **Empírico Experimental**, debido a que la investigación se va a realizar para el esclarecimiento de las propiedades, relaciones y leyes del objeto de estudio, así como para verificar la hipótesis. El experimento tendrá carácter transformador, pues se trata de revelar la realidad y se actuar sobre ella para transformarla, ejemplo de ello es al identificar de los elementos significativos y generalizables, desde el punto de vista arquitectónico de las herramientas (CASE), existentes hoy en el entorno productivo cubano, se podrá transformar la misma con elementos nuevos e incorporar los ya existentes, para integrarlos en la definición de la arquitectura una nueva herramienta CASE a realizar en nuestra universidad, cuyo nombre es Herramienta para la Modelación de Aplicaciones Educativas: ApEM-H ().

1.3 Herramientas CASE.

Herramientas CASE (Ingeniería de Software Asistida por Computadora): son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas

herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Se define también como:

Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.

La sigla genérica para una serie de programas y una filosofía de desarrollo de software que ayuda a automatizar el ciclo de vida de desarrollo de los sistemas.

Una innovación en la organización, un concepto avanzado en la evolución de tecnología con un potencial efecto profundo en la organización. Se puede ver al CASE como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

[Ver **Anexo 1**]. Variaciones en el significado de CASE (**Imagen 1**).

Ventajas.

- Incremento en la velocidad de desarrollo de los sistemas.
- Permite desarrollar sistemas sin encarar el problema de tener cambios en las necesidades del negocio, antes de finalizar el proceso de desarrollo.
- Permite a las compañías competir más efectivamente usando estos sistemas desarrollados nuevamente para compararlos con sus necesidades de negocio actuales.
- Permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar.
- Permite a las organizaciones desarrollar rápidamente sistemas de mejor calidad para soportar procesos críticos del negocio y asistir en el desarrollo y promoción intensiva de la información de productos y servicios.
- Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunas de ellas son:
- Verificar el uso de todos los elementos en el sistema diseñado.

- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la Base de Datos.
- Generar estructuras de código.
- Mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad.
- Mejora en la calidad, fiabilidad, utilidad y rendimiento.
- El entorno de producción de documentación para software mejora la comunicación, mantenimiento y actualización.
- Hace el trabajo de diseño de software más fácil y agradable.
- Reducción del costo de producción de software.

1.3.1 Clasificación de las Herramientas CASE.

No existe una única clasificación de herramientas CASE y, en ocasiones, es difícil incluirlas en una clase determinada. Podrían clasificarse atendiendo a:

1. Las plataformas que soportan.
2. La arquitectura de las aplicaciones que producen.
3. Las fases del ciclo de vida del desarrollo de sistemas que cubren:

    a) Herramientas integradas, I-CASE (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.

    b) Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o front-end, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño.

    c) Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o back-end, dirigidas a las últimas fases del desarrollo: construcción e implantación.

    d) Juegos de herramientas o Tools-Case, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se

encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento [3].

4. Su funcionalidad.

   a) Herramientas de planificación de sistemas de gestión. Sirven para modelizar los requisitos de información estratégica de una organización. Proporcionan un "meta modelo" del cual se pueden obtener sistemas de información específicos. Su objetivo principal es ayudar a comprender mejor cómo se mueve la información entre las distintas unidades organizativas. Estas herramientas proporcionan una ayuda importante cuando se diseñan nuevas estrategias para los sistemas de información y cuando los métodos y sistemas actuales no satisfacen las necesidades de la organización.

   b) Herramientas de análisis y diseño. Permiten al desarrollador crear un modelo del sistema que se va a construir y también la evaluación de la validez y consistencia de este modelo. Proporcionan un grado de confianza en la representación del análisis y ayudan a eliminar errores con anticipación.

      • Herramientas de análisis y diseño (Modelamiento).

      • Herramientas de creación de prototipos y de simulación.

      • Herramientas para el diseño y desarrollo de interfaces.

      • Máquinas de análisis y diseño (Modelamiento).

   c) Herramientas de programación. Se engloban aquí los compiladores, los editores y los depuradores de los lenguajes de programación convencionales.

   Ejemplos de estas herramientas son:

   • Herramientas de codificación convencionales.

   • Herramientas de codificación de cuarta generación.

   • Herramientas de programación orientadas a los objetos.

   d) Herramientas de integración y prueba: Sirven de ayuda a la adquisición, medición, simulación y prueba de los equipos lógicos desarrollados. Entre las más utilizadas están:

• Herramientas de análisis estático.

• Herramientas de codificación de cuarta generación.

• Herramientas de programación orientadas a los objetos.

e) Herramientas de gestión de prototipos. Los prototipos son utilizados ampliamente en el desarrollo de aplicaciones, para la evaluación de especificaciones de un sistema de información, o para un mejor entendimiento de cómo los requisitos de un sistema de información se ajustan a los objetivos perseguidos.

f) Herramientas de mantenimiento: La categoría de herramientas de mantenimiento se puede subdividir en:

• Herramientas de ingeniería inversa.

• Herramientas de reestructuración y análisis de código.

• Herramientas de reingeniería.

g) Herramientas de gestión de proyectos. La mayoría de las herramientas CASE de gestión de proyectos se centran en un elemento específico de la gestión del proyecto, en lugar de proporcionar un soporte global para la actividad de gestión. Utilizando un conjunto seleccionado de las mismas se puede: realizar estimaciones de esfuerzo, coste y duración, hacer un seguimiento continuo del proyecto, estimar la productividad y la calidad, etc. Existen también herramientas que permiten al comprador del desarrollo de un sistema, hacer un seguimiento que va desde los requisitos del pliego de prescripciones técnicas inicial, hasta el trabajo de desarrollo que convierte estos requisitos en un producto final. Se incluyen dentro de las herramientas de control de proyectos las siguientes:

• Herramientas de planificación de proyectos.

• Herramientas de seguimiento de requisitos.

• Herramientas de gestión y medida.

h) Herramientas de soporte. Se engloban en esta categoría las herramientas que recogen las actividades aplicables en todo el proceso de desarrollo, como las que se relacionan a continuación:

- Herramientas de documentación.
- Herramientas para software de sistemas.
- Herramientas de control de calidad.
- Herramientas de bases de datos.

Otra clasificación, diferencia las funciones CASE en cinco grupos:

1. Repositorio.

Funcionan en torno a un repositorio central, siendo éste el núcleo fundamental que contiene todas las definiciones de objeto y sus relaciones. Los objetos pueden ser especificaciones del sistema en forma de diagramas de flujo de datos, diagramas entidad-relación, esquemas de bases de datos, diseños de pantallas, etc. El repositorio es un concepto más amplio que el de diccionario de datos y soporta a los demás grupos de funciones. No es fácil encontrar en el mercado productos CASE con funcionalidades estrictamente a las de repositorio, ya que, a pesar de su innegable importancia, tienen un carácter auxiliar de los demás grupos de funciones. Cualquier sistema CASE poseerá un repositorio propio o bien, trabajará sobre un repositorio suministrado por otro fabricante o vendedor [4].

2. Re-ingeniería.

Los sistemas CASE permiten establecer una relación estrecha y fuertemente formalizable entre los productos generados a lo largo de distintas fases del ciclo de vida, permitiendo actuar en el sentido especificaciones-código (ingeniería "directa") y también en el contrario (ingeniería "inversa"). Ello facilita la realización de modificaciones en la fase más adecuada en cada caso y su traslado a las demás. Al conjunto de facilidades proporcionadas por la ingeniería «directa» e "inversa" se le denomina "re-ingeniería".

3. Soporte del ciclo de vida.

El ciclo de vida de una aplicación o de un sistema de información se compone de varias etapas, que van desde la planificación de su desarrollo hasta su implantación, mantenimiento y actualización. Aunque el número de fases puede ser variable en función del nivel de detalle que se adopte, pueden de modo simplificado, identificarse las siguientes:

- Planeamiento.
- Análisis y Diseño.

- Implantación (programación y pruebas).
- Mantenimiento y actualización.

Los sistemas CASE pueden cubrir la totalidad de estas fases o bien especializarse en algunas de ellas. En este último caso se pueden distinguir sistemas de "alto nivel" ("Upper CASE"), orientados a la autonomía y soporte de las actividades correspondientes a las dos primeras fases y, sistemas de "bajo nivel" ("Lower CASE"), dirigidos hacia las dos últimas. Los sistemas de "alto nivel" pueden soportar un número más o menos amplio de metodologías de desarrollo [4].

4. Soporte de proyecto.

Este tipo de funciones hace referencia al soporte de actividades que se producen durante el desarrollo, derivadas fundamentalmente del trabajo en grupos, tales como facilidades de comunicación, soporte a la creación, modificación e intercambio de documentación, herramientas personales, controles de seguridad, etc. Los sistemas CASE pueden conceder a estas cuestiones una importancia variable por lo cual el soporte de proyecto constituye un factor de diferenciación.

5. Mejora continua de calidad.

Aunque frecuentemente se asocia a los sistemas CASE con la mejora de la productividad en el desarrollo de aplicaciones, debe tenerse en cuenta que una de las principales ventajas estriba también, en la mejora de la calidad de los desarrollos realizados. Determinados sistemas CASE enfatizan más sobre este punto que sobre el anterior, introduciendo herramientas que permiten ejercer un control intenso de garantía de calidad del software desarrollado desde las primeras fases de su ciclo de vida [5].

1.3.2 Rango de las Herramientas Case.

Algunas Herramientas CASE son sólo para la fase de Diseño.

Otras, son sólo generadoras de Código.

Algunas Herramientas de Análisis y Diseño tienen una visión de Desarrollo orientada a procesos sin la capacidad de modelamiento.

Algunas proveen Herramientas para el modelamiento sin incluir los procesos de Análisis o Diseño.

1.3.3 Características deseables de una CASE.

Una herramienta CASE cliente / servidor provee modelo de datos, generación de código, registro del ciclo de vida de los proyectos, comunicación entre distintos ingenieros. Las principales herramientas son KnowledgeWare's Application Development Workbench, TI's, Information Engineering Facility (IEF), y Andersen consulting's Foundation for Cooperative Processing.

Deberes de una herramienta CASE Cliente / servidor:

- Proporcionar topologías de aplicación flexibles. La herramienta debe proporcionar facilidades de construcción que permita separar la aplicación (en muchos puntos diferentes) entre el cliente, el servidor y más importante, entre servidores.
- Proporcionar aplicaciones portátiles. La herramienta debe generar código para Windows, OS/ 2, Macintosh, Unix y todas las plataformas de servidores conocidas. Debe ser capaz, a tiempo de corrida, desplegar la versión correcta del código en la máquina apropiada.
- Control de Versión. La herramienta debe reconocer las versiones de códigos que se ejecutan en los clientes y servidores, y asegurarse que sean consistentes. También,

la herramienta debe ser capaz de controlar un gran número de tipos de objetos incluyendo texto, gráficos, mapas de bits, documentos complejos y objetos únicos, tales como definiciones de pantallas y de informes, archivos de objetos y datos de prueba y resultados. Debe mantener versiones de objetos con niveles arbitrarios de granularidad; por ejemplo, una única definición de datos o una agrupación de módulos.

- Crear código compilado en el servidor. La herramienta debe ser capaz de compilar automáticamente código 4GL en el servidor para obtener el máximo performance.

- Trabajar con una variedad de administradores de recurso. La herramienta debe adaptarse ella misma a los administradores de recurso que existen en varios servidores de la red; su interacción con los administradores de recurso debería ser negociable a tiempo de ejecución.

- Trabajar con una variedad de software intermedio. La herramienta debe adaptar sus comunicaciones cliente / servidor al software intermedio existente. Como mínimo la herramienta debería ajustar los temporizadores basándose en, si el tráfico se está moviendo en una LAN o WAN.

- Soporte multiusuarios. La herramienta debe permitir que varios diseñadores trabajen en una aplicación simultáneamente. Debe gestionarse los accesos concurrentes a la base de datos por diferentes usuarios, mediante el arbitrio y bloqueos de accesos a nivel de archivo o de registro.

- Seguridad. La herramienta debe proporcionar mecanismos para controlar el acceso y las modificaciones a los que contiene. La herramienta debe, al menos, mantener contraseñas y permisos de acceso en distintos niveles para cada usuario. También debe facilitar la realización automática de copias de seguridad y recuperaciones de las mismas, así como el almacenamiento de grupos de información determinados, por ejemplo, por proyecto o aplicaciones.

- Desarrollo en equipo, repositorio de librerías compartidas. Debe permitir que grupos de programadores trabajen en un proyecto común; debe proveer facilidades de check-in/ check-out registrar formas, widgets, controles, campos, objetos de negocio, DLL, etc.; debe proporcionar un mecanismo para compartir las librerías entre distintos realizadores y múltiples herramientas; Gestiona y controla el acceso multiusuario a los datos y bloquea los objetos para evitar que se pierdan modificaciones inadvertidamente cuando se realizan simultáneamente [4].

1.3.4 Factores asociados a la implantación de las herramientas CASE.

La difusión de las innovaciones en esta área ha comenzado a estudiarse a partir de los años 1940. Por ello, existen estudios teóricos al respecto, realizándose evaluaciones, adopción e implementación tecnológica.

Existe un amplio cuerpo de investigaciones disponibles sobre la adopción de innovaciones. Muchos de los estudios sobre innovación se han analizado bajo dos perspectivas: adopción y difusión (Kimberly, 1981). Mientras unos estudios usan la perspectiva de la adopción para evaluar la receptividad y los cambios de la organización o sociedad por la innovación, otros usan la perspectiva de la difusión para intentar entender por qué y cómo se difunde y qué características generales o principales de la innovación son aceptadas.

1.3.5 Sobre las "I – CASE" (Herramientas CASE Integradas).

El I-CASE se concibe como el conjunto de cuatro herramientas que tocan las disciplinas que van desde la estrategia de la empresa, y la concepción del sistema de información, hasta el análisis, diseño y la generación de los mismos programas. Las herramientas I-CASE se basan en una metodología. Tienen un repositorio y aportan técnicas estructuradas para todas las fases del ciclo de vida. Estas son las características que les confieren su mayor ventaja: una mejora de la calidad de los desarrollos.

[Ver **Anexo 2**]. Comparación entre el desarrollo tradicional de software, desarrollo utilizando una herramienta CASE y una herramienta I-CASE (**Tabla 1**).

Como se ha dicho con anterioridad en este documento, las aplicaciones CASE tienen un gran potencial para ayudar a los encargados del desarrollo de software a realizar sus tareas de una manera más automatizada y eficiente. Pero para poder obtener de manera óptima las ventajas de ellas lo ideal es tenerlas en un Ambiente Integrado, donde no se desperdicie el esfuerzo en traspasar la información de un proyecto desde una fase de desarrollo de software a la siguiente [4].

Hay todo un espectro de tipos de integración que puede tener un grupo de herramientas CASE, y para que se puedan considerar como realmente integradas, deben estar en el último nivel de este rango. Así, para explicar comprensiblemente lo que es un Ambiente Integrado de herramientas CASE (I-CASE) o un Entorno de Apoyo de Proyectos Integrado (EAPI) debemos entender lo que no lo es para poder diferenciarlos. A continuación se describirán los distintos tipos de integración, desde el nulo hasta un EAPI [6].

Herramienta Individual: Su nombre las describe, pues son las herramientas que se utilizan exclusivamente para una fase del desarrollo de software. No tienen manera de recibir información más que la que proporcione el usuario al momento de utilizarlas, y la que den los proyectos creados anteriormente con tal herramienta [6].

Intercambio de Datos: Este se realiza cuando las herramientas guardan su información en ciertos formatos que podrían ser leídos por otras herramientas CASE que acepten dicho formato. Este intercambio de datos se puede ver con una flecha, ya sea unidireccional o bidireccional.

Puente: Estos se crean cuando hay intercambios de datos específicos para herramientas que se complementan por cubrir fases consecutivas en el desarrollo de software. También se utilizan los puentes para todo un grupo o serie de herramientas, en lo que se denomina un consorcio de estándares [6].

Fuente Única: Esta es una suite (conjunto de programas) de aplicaciones CASE de un mismo proveedor. La ventaja de este enfoque es que el traspaso de información suele ser transparente entre las herramientas. Desafortunadamente por lo general solamente pueden interactuar entre ellas, y no es posible añadir otra herramienta para complementar las necesidades específicas del usuario [6].

EAPI: El Entorno de Apoyo a Proyectos Integrado se construye mediante estándares de traspaso de información y meta datos alrededor de un depósito de datos, lo cual permite añadir herramientas que se ajusten a los estándares, aunque sean de distintos proveedores, con las ventajas que proporcionan las bases de datos.

Como se puede ver, un Entorno Integrado provee una simplificación en la transferencia de datos entre herramientas, con una consecuente mejora en el proceso del flujo de información. Esto por consecuencia nos brinda una reducción en el esfuerzo para realizar actividades de control de los proyectos, equipos y software, e incluso nos proporciona mejores control y coordinación de las actividades del personal de desarrollo [6].

1.3.6 Lista de algunas aplicaciones CASE:

AllFusion ERWin, ArgoUML, Blue Ink, BPWin , CASE Studio 2, CASEWise , Database Designer for MySQL, DBDesigner 4, DMS Software Reengineering Toolkit, EasyCase, Eclipse, Embarcadero ER/Studio, Enterprise Architect, eREQUIREMENTS - eRequirements, GeneXus, GNU Ferret, INNOVATOR, iRise, IRqA, MagicDraw, MetaCASE, Modelistic, Obsydian / Plex , Oracle Designer , Rational ClearCASE, Rational Rose, SILVERRUN,

swREUSER, Sybase PowerDesigner, System Architect, Together, Topcased, Umbrello, Visible Enterprise Products, Visual Paradigm for UML, Xcase Database Design Software.

1.3.7 Herramientas CASE utilizadas en el ámbito internacional.

ERwin

PLATINUM ERwin es una herramienta de diseño de base de datos. Brinda productividad en diseño, generación, y mantenimiento de aplicaciones. Desde un modelo lógico de los requerimientos de información, hasta el modelo físico perfeccionado para las características específicas de la base de datos diseñada, ERwin permite visualizar la estructura, los elementos importantes, y optimizar el diseño de la base de datos. Genera automáticamente las tablas y miles de líneas de stored procedure y triggers para los principales tipos de base de datos.

ERwin establece una conexión entre una base de datos diseñada y una base de datos, permitiendo transferencia entre ambas y la aplicación de ingeniería reversa. Usando esta conexión, Erwin genera automáticamente tablas, vistas, índices, reglas de integridad referencial (llaves primarias, llaves foráneas), valores por defecto y restricciones de campos y dominios. ERwin soporta principalmente bases de datos relacionales SQL y bases de datos que incluyen Oracle, Microsoft SQL Server, Sybase, DB2, e Informix. El mismo modelo puede ser usado para generar múltiples bases de datos, o convertir una aplicación de una plataforma de base de datos a otra.

System Architect

System Architect posee un repositorio único que integra todas las herramientas, y metodologías usadas. En la elaboración de los diagramas, el System Architect conecta directamente al diccionario de datos, los elementos asociados, comentarios, reglas de validaciones, normalización, etc. Posee control automático de diagramas y datos, normalizaciones y balanceamiento entre diagramas "Padre e Hijo", además de balanceamiento horizontal, que trabaja integrado con el diccionario de datos, asegurando la compatibilidad entre el Modelo de Datos y el Modelo Funcional.

System Architect es considerado un Upper CASE, que puede ser integrado a la mayoría de los generadores de código. Traduce modelos de entidades, a partir de la enciclopedia, en esquemas para Sybase, DB2, Oracle u Oracle 7, Ingress, SQL Server, RDB, XDB, Progress, Paradox, SQL Base, AS400, Interbase, OS/2, DBMS, Dbase 111, Informix, entre otros. Genera también Windows DDL, definiciones de datos para lenguaje C/C++ y estructuras de datos en Cobol. En esta ultima versión del System Architect es posible a

través de ODBC, la creación de bases de datos a partir del modelo de entidades, para los diversos manejadores de bases de datos arriba mencionados. Posee esquemas de seguridad e integridad a través de contraseñas que posibilitan el acceso al sistema en diversos niveles, pudiéndose integrar a la seguridad de la red Novell o Windows/NT de ser necesario. Posee también con un completo Help sensible al contexto. System Architect posee un módulo específico para Ingeniería Reversa desde las Bases de Datos SQL más populares, incluyendo Sybase, DB2, Infonmix, Oracle y SQL Server (DLL), además de diálogos (DLG) y menús (MNU) desde Windows.

1.3.8 Herramientas CASE más utilizadas en el ámbito nacional.

RATIONAL ROSE.

Justo en el momento que las organizaciones tienen que crear software de mayor calidad, están bajo la presión de trabajar más rápido que nunca antes. Porque el desarrollo cada vez más se realiza a velocidad "Internet", los ciclos de desarrollo se están volviendo mucho más cortos.

En el pasado, las compañías podían escoger sacrificar la calidad del software por una entrega a tiempo, o descartar características del software con el fin de cumplir los tiempos de entrega al mercado. En la nueva economía de Internet, ninguna de las dos opciones es posible. Las compañías deben producir software de mayor calidad en mucho menos tiempo. Punto.

Rational Rose le permite visualizar, entender, y refinar sus requerimientos y arquitectura antes de enfrentarse al código. Esto le permite evitar esfuerzos desperdiciados en el ciclo de desarrollo. Usar una sola herramienta de modelamiento a través del ciclo de vida del desarrollo le permite asegurar que usted está construyendo el sistema correcto. El modelo arquitectónico puede ser rastreado hacia el modelo de procesos de negocios y los requerimientos de sistema.

VISUAL PARADIGM

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Lista de características:

- Soporte de UML versión 2.1
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento
- Modelado colaborativo con CVS y Subversión (nueva característica)
- Interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XMI (nueva característica).
- Ingeniería de ida y vuelta
- Ingeniería inversa - Código a modelo, código a diagrama
- Ingeniería inversa Java, C++, Esquemas XML, XML,.NET exe/dll, CORBA IDL
- Generación de código - Modelo a código, diagrama a código
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso
- Diagramas EJB - Visualización de sistemas EJB.
- Generación de código y despliegue de EJB´s - Generación de beans para el desarrollo y despliegue de aplicaciones.
- Diagramas de flujo de datos
- Soporte ORM - Generación de objetos Java desde la base de datos
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación
- Generador de informes para generación de documentación
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML
- Importación y exportación de ficheros XMI

- Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio
- Editor de figuras.

1.4 Conclusiones

En este capítulo se hizo la descripción de las herramientas CASE más utilizadas para prestar servicios orientados a el soporte y automatización de las diferentes actividades y tareas que se desarrollan durante el ciclo de vida del software durante sus etapas consecutivas, como es el caso de Rational Rose de IBM PC, Visual Paradigm, System Architect, ERwin etc., así como el rango que cubren las herramientas CASE y las características deseables de una herramienta CASE y clasificaciones de las herramientas, haciendo énfasis en las I-CASE, que es el rango o nivel abarcador de estas herramientas y que será el escogido para que ApEM-H pertenezca al mismo.

CAPÍTULO 2

ARQUITECTURA DE SOFTWARE. ARQUITECTURA DE INTEGRACIÓN CASE.

2.1 Introducción

En este capítulo se realizará una descripción de diferentes conceptos establecidos referente a la arquitectura de software, así como una descripción de los diferentes modelos de arquitectura que existen, resaltando la arquitectura en 3 capas así como los diferentes patrones arquitectónicos que existen, resaltando el patrón MVC (Modelo Vista Controlador), así como las modalidades y tendencias de la arquitectura de software, así como los modelos arquitectónicos definidos específicamente para la construcción de ambientes CASE integrados, como es el caso del modelo de arquitectura definido por Pressman, también conocido como marco de referencia de integración, para la construcción de un entorno o ambiente CASE OO (Orientado a Objeto).

2.2 Arquitectura de Software.

2.2.1 Definición de Arquitectura de Software.

Una definición reconocida es la de Clements: la AS (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.

Un componente es una cosa, una entidad, a la que los arquitectos prefieren llamar “componente” antes que “objeto”. La AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño.

La Arquitectura de Software constituye la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

2.2.2 Modelos de Arquitectura.

1) Modelos estructurales: Sostienen que la AS está compuesta por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizada 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.

2.1.3 Lenguajes de Descripción Arquitectónicos (ADLs)

Una ves que el arquitecto se decide a delinear su estrategia para proponer la arquitectura de un sistema, debe tener en cuenta que utilizará para modelar la misma de forma visual y que pueda tener un alto nivel de abstracción. Los ADLs se utilizan, además, para satisfacer requerimientos descriptivos de alto nivel de abstracción que las herramientas basadas en objeto en general y UML en particular no cumplen satisfactoriamente. Aunque algunos arquitectos alegan que el período de gloria de la OOP podría estar acercándose a su fin, el hecho concreto es que el modelado orientado a objetos de sistemas basados en

componentes posee cierto número de rasgos muy convenientes a la hora de diseñar o al menos describir un sistema. En primer lugar, las notaciones de objeto proporcionan un mapeo directo entre un modelo y una implementación e implementan además métodos bien definidos para desarrollar sistemas a partir de un conjunto de requerimientos.

Contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico. Algunas de esas propiedades podrían ser, por ejemplo, protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de una posible evolución del sistema. Entre los ADL más difundidos podemos encontrar:

Acme – Armani

Acme se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADLs, o en otras palabras, como un lenguaje de intercambio de arquitectura. No es entonces considerado por muchos un ADL en sentido estricto. De hecho, posee numerosas prestaciones que también son propias de los ADLs. En su sitio oficial se reconoce que como ADL no es necesariamente apto para cualquier clase de sistemas, al mismo tiempo que se destaca su capacidad de describir con facilidad sistemas “relativamente simples”.

Aesop

El nombre oficial es Aesop Software Architecture Design Environment Generator. Se ha desarrollado como parte del proyecto ABLE de la Universidad Carnegie Mellon, cuyo objetivo es la exploración de las bases formales de la arquitectura de software, el desarrollo del concepto de estilo arquitectónico y la producción de herramientas útiles a la arquitectura, de las cuales Aesop es precisamente la más relevante. Aesop genera código C++ y aunque opera primariamente desde una interfaz visual, el código de es marcadamente más procedural que el de Acme. Disponibilidad de plataforma – Aesop no está disponible en plataforma Windows, aunque naturalmente puede utilizarse para modelar sistemas implementados en cualquier plataforma.

ArTek

ArTek fue desarrollado por Teknowledge. Se lo conoce también como ARDEC/Teknowledge Architecture Description Language. Para algunos arquitectos no es considerado un genuino ADL, por cuanto la configuración es modelada implícitamente mediante información de

interconexión que se distribuye entre la definición de los componentes individuales y los conectores. Aunque pueda no ser un ADL en sentido estricto, se le reconoce la capacidad de modelar ciertos aspectos de una arquitectura. Disponibilidad de plataforma – Hoy en día ArTek no se encuentra disponible en ningún sitio y para ninguna plataforma

Darwin

Darwin es un lenguaje de descripción arquitectónica desarrollado por Jeff Magee y Jeff Kramer. El mismo describe un tipo de componente mediante una interfaz consistente en una colección de servicios que son ya sea provistos (declarados por ese componente) o requeridos (o sea, que se espera ocurran en el entorno). Las configuraciones se desarrollan instanciando las declaraciones de componentes y estableciendo vínculos entre ambas clases de servicios. En una implementación generada en Darwin, se supone que cada componente primitivo está implementado en algún lenguaje de programación, y que para cada tipo de servicio se necesita un ligamento (glue) que depende de cada plataforma. El algoritmo de elaboración actúa, esencialmente, como un servidor de nombre que proporciona la ubicación de los servicios provistos a cualquier componente que se ejecute.

Disponibilidad de plataforma – Aunque el ADL fue originalmente planeado para ambientes tan poco vinculados al modelado corporativo como hoy en día lo es Macintosh, en Windows se puede modelar en lenguaje Darwin.

Jacal

Es un lenguaje de descripción de arquitecturas de software de propósito general creado en la Universidad de Buenos Aires, por un grupo de investigación del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales.

El objetivo principal de Jacal es lo que actualmente se denomina “animación” de arquitecturas. Esto es, poder visualizar una simulación de cómo se comportaría en la práctica un sistema basado en la arquitectura que se ha representado. Más allá de este objetivo principal, el diseño de Jacal contempla otras características deseables en un ADL, como por ejemplo contar con una representación gráfica que permita a simple vista transmitir la arquitectura del sistema, sin necesidad de recurrir a información adicional. Para este fin, se cuenta con un conjunto predefinido (extensible) de conectores, cada uno con una representación distinta.

UML

UML forma parte del repertorio conocido como lenguajes semi-formales de modelado. En términos de número, muchos de los conocedores de UML no admite comparación con la todavía modesta población de especialistas en ADLs. Existen dos fracciones claramente definidas; la primera, vinculada con el mundo de Rational y UML, impulsa el uso casi irrestricto de UML como si fuera un ADL normal; la segunda ha señalado reiteradas veces las limitaciones de UML no sólo como ADL sino como lenguaje universal de modelado.

2.1.4 Modalidades y Tendencias de la Arquitectura de Software

1) Arquitectura como etapa de ingeniería y diseño orientada a objetos**:** Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de UML y Rational. No cabe duda que se trata de una corriente específica: Rumbaugh, Jacobson y Booch han sido llamados “Los Tres Amigos”; de lo que sí puede dudarse es que se trate de una postura arquitectónica. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción, pero no está sistemáticamente ligada al requerimiento que viene antes o a la composición del diseño que viene después. Lo que sigue al momento arquitectónico es business as usual, y a cualquier configuración, topología o morfología de las piezas del sistema se la llama arquitectura. En esta escuela, si bien se reconoce el valor primordial de la abstracción (nadie después de Dijkstra osaría oponerse a ello) y del ocultamiento de información promovido por Parnas, estos conceptos tienen que ver más con el encapsulamiento en clases y objetos que con la visión de conjunto arquitectónica. Para este movimiento, la arquitectura se confunde también con el modelado y el diseño, los cuales constituyen los conceptos dominantes. En esta corriente se manifiesta predilección por un modelado denso y una profusión de diagramas, tendiente al modelo metodológico CMM o a UPM. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. Cuando aquí se habla de estilos, se los confunde con patrones arquitectónicos o de diseño. Jamás se hace referencia a los lenguajes de descripción arquitectónica, que representan uno de los assets reconocidos de la AS; sucede como si la disponibilidad de un lenguaje unificado de modelado los tornara superfluos. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo; esta arquitectura es isomorfa a la estructura de las piezas de código. Una definición típica y demostrativa sería la de Grady Booch; para él, la AS es “la estructura lógica y física de un sistema, forjada por todas las decisiones

estratégicas y tácticas que se aplican durante el desarrollo". Otras 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 descriptas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten.

2) 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. Se trata también de la visión de la AS dominante en la academia, y aunque es la que ha hecho el esfuerzo más importante por el reconocimiento de la AS como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial. En el interior del movimiento se pueden observar distintas divisiones. Hay una variante informal de "boxología" y una vertiente más formalista, representada por el grupo de Mark Moriconi en el SRI de Menlo Park. En principio se pueden reconocer tres modalidades en cuanto a la formalización; los más informales utilizan descripciones verbales o diagramas de cajas, los de talante intermedio se sirven de ADLs y los más exigentes usan lenguajes formales de especificación como CHAM y Z. 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, y en general nadie habla de clases o de objetos. Mientras algunos participantes excluyen el modelo de datos de las incumbencias de la AS (Shaw, Garlan, etc), otros insisten en su relevancia (Medvidovic, Taylor). Todo estructuralismo es estático; hasta fines del siglo XX, no está muy claro el tema de la posición del modelado arquitectónico en el ciclo de vida.

3) Estructuralismo arquitectónico radical: Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más confrontativa con el mundo UML. En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la AS y otra que procura definir nuevos meta modelos y estereotipos de UML como correctivos de la situación.

4) Arquitectura basada en patrones: Si bien reconoce la importancia de un modelo emanado históricamente del diseño orientado a objetos, esta corriente surgida hacia

1996 no se encuentra tan rígidamente vinculada a UML en el modelado, ni a CMM en la metodología. El texto sobre patrones que esta variante reconoce como referencia es la serie POSA de Buschmann y otros y secundariamente el texto de la Banda de los Cuatro. La diferencia entre ambos textos sagrados de la comunidad de patrones no es menor; en el primero, la expresión "Software Architecture" figura en el mismo título; el segundo se llama Design Patterns: Elements of reusable Object-Oriented software y su tratamiento de la arquitectura es mínimo. En esta manifestación de la AS prevalece cierta tolerancia hacia modelos de proceso tácticos, no tan macroscópicos, y eventualmente se expresa cierta simpatía por las ideas de Martin Fowler y las premisas de la programación extrema. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura.

5) Arquitectura procesual: Desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos (no todos) los 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, Charles Weinstock. Intenta establecer modelos de ciclo de vida y técnicas de elicitación de requerimientos, brainstorming, diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software. Toda la documentación puede encontrarse ordenada en el SEI, pero no se mezcla jamás con la de CMM, a la que redefine de punta a punta. Otras variantes dentro de la corriente procesual caracterizan de otras maneras de etapas del proceso: extracción de arquitectura, generalización, reutilización.

6) Arquitectura basada en escenarios: Es la corriente más nueva. Se trata de un movimiento predominantemente europeo, con centro en Holanda. Recupera el nexo de la AS con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. Los teóricos y practicantes de esta modalidad de arquitectura se inscriben dentro del canon delineado por la arquitectura procesual, respecto de la cual el movimiento constituye una especialización. En esta corriente suele utilizarse diagramas de casos de uso UML como herramienta informal u ocasional, dado que los casos de uso son uno de los escenarios posibles. Los casos de uso no están orientados a objeto. Los autores vinculados con esta modalidad han sido, aparte de los codificadores de ATAM, CBAM, QASAR y demás métodos del SEI, los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research: Mugurel Ionita, Dieter Hammer, Henk Obbink, Hans de Bruin, Hans van Vliet, Eelke Folmer, Jilles van Gurp,

Jan Bosch. La profusión de holandeses es significativa; la Universidad de Eindhoven es, incidentalmente, el lugar en el que surgió lo que P. I. Sharp proponía llamar la escuela arquitectónica de Dijkstra.

2.1.5 Patrones y estilos arquitectónicos.

Como conceptos, los estilos fueron formulados por primera vez cuando el escenario tecnológico era sustancialmente distinto del que se manifiesta hoy en día. Es por eso que en este estudio se analizarán las definiciones de los estilos arquitectónicos que se han propuesto, así como su posición en el campo de las vistas de arquitectura, su lugar en la metodología y su relación con los patrones, tomando en consideración las innovaciones experimentadas por el campo de los estilos desde su formulación inicial en 1992, coincidente con el instante que la IEEE identifica como el del nacimiento de la arquitectura de software en sentido estricto. Desde que surgieran tanto la disciplina como los estilos no sólo se han generalizado arquitecturas de cliente-servidor, sino que experimentaron su auge primero las configuraciones en varias capas y luego las basadas en componentes, en recursos (la Web) y en servicios (los Web services). Las metodologías de ingeniería también experimentaron evoluciones naturales y hasta revoluciones extremas, de modo que habrá que analizar en algún momento si ciertas prácticas que antes se daban por sentadas siguen o no en pie y en qué estado de salud se encuentran; y como el trabajo que se está leyendo se realiza en un momento en que son unos cuantos los que hablan de crisis, habrá que ver si las ideas en juego tienden a acentuar los problemas o constituyen una vía de solución.

Los estilos sólo se manifiestan en arquitectura teórica descriptiva de alto nivel de abstracción; los patrones, por todas partes. Los partidarios de los estilos se definen desde el inicio como arquitectos; los que se agrupan en torno de los patrones se confunden a veces con ingenieros y diseñadores, cuando no con programadores con conciencia sistemática o lo que alguna vez se llamó analistas de software. El primer grupo ha abundado en taxonomías internas de los estilos y en reflexión teórica; el segundo se ha mantenido, en general, refractario al impulso taxonómico, llevado por una actitud resueltamente empírica. Ambos, con mayor o menor plenitud y autoconciencia, participan del campo abarcativo de la arquitectura de software. Los estilos se encuentran en el centro de la arquitectura y constituyen buena parte de su sustancia. Los patrones de arquitectura están claramente dentro de la disciplina arquitectónica, solapándose con los estilos, mientras que los patrones de diseño se encuentran más bien en la periferia, si es que no decididamente afuera.

Los estilos que habrán de describirse a continuación no aspiran a ser todos los que se han propuesto, sino apenas los más representativos y vigentes. De más está decir que, como se ha visto, la agrupación de estilos y sub-estilos es susceptible de realizarse de múltiples formas, conforme a los criterios que se apliquen en la constitución de los ejemplares. No se ha de rendir cuentas aquí por la congruencia de la clasificación (nadie ha hecho lo propio con las suyas), porque cada vez que se la ha revisado en modo outline, se cedió a la tentación de cambiar su estructura, la cual seguirá sufriendo metamorfosis después de despachado el artículo. Se podrá apreciar que, en consonancia con los usos de la especialidad, la caracterización de los estilos no constituye un reflejo pormenorizado de los detalles de sus estructuras, sino una visión deliberadamente sucinta y genérica que destaca sus valores esenciales y sus rasgos distintivos.

Los patrones de arquitectura 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. Los patrones de arquitectura representan el nivel más alto en el sistema.

1) Patrón Modelo Vista Controlador (MVC)

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

Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y mostrarlos al usuario. Luego que el usuario introduce modificaciones, las mismas se reflejan en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) es unir ambas piezas para reducir la cantidad de código y optimizar la performance. Sin embargo, esta idea es antagónica al hecho de que la interfaz suele cambiar, o acostumbra depender de distintas clases de dispositivos (clientes ricos, browsers, PDAs); la programación de interfaces de HTML, además, requiere habilidades muy distintas de la programación de lógica de negocios. Otro problema es que las aplicaciones tienden a incorporar lógica de negocios que van más allá de la transmisión de datos.

[Ver **Anexo 1**]. Modelo – Vista – Controlador (**Imagen 2**).

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Maneja la visualización de la información.

Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

- Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual. La separación entre vista y controlador puede ser secundaria en aplicaciones de clientes ricos y, de hecho, muchos frameworks de interfaz implementan ambos roles en un solo objeto. En aplicaciones de Web, por otra parte, la separación entre la vista (el browser) y el controlador (los componentes del lado del servidor que manejan los requerimientos de HTTP) está mucho más taxativamente definida.

Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una lluvia de requerimientos de actualización. Hace pocos años sucedía que algunas vistas, tales como las pantallas gráficas, involucraban más tiempo para plasmar el dibujo que el que demandaban los nuevos requerimientos de actualización.

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 (por ejemplo, el controlador actualiza el carro de la compra del usuario).

4. Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

5. 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 (por ejemplo, produce un listado del contenido del carro de la compra). 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.

6. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente

   Entre las ventajas del patrón podemos apreciar:

- Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación de Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.
- Adaptación al cambio. Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo. Este patrón sentó las bases para especializaciones ulteriores, tales como Page Controller y Front Controller.

2) Patrón Capas

Con anterioridad se había explicado como uno de los estilos arquitectónicos, ahora se hará referencia a cada uno de los ejemplos específicos que pueden aparecer de este patrón partiendo que la funcionalidad principal del mismo es crear una modularidad del sistema asignando a cada capa funcionalidades específicas y bien definidas.

La capa de la Presentación

Esta capa reúne todos los aspectos del software que tiene que ver con las interfaces y la interacción con los diferentes tipos de usuarios. Estos típicamente incluyen el manejo y aspecto de las ventanas, el formato de los reportes, menús, gráficos y elementos multimedia en general.

La capa del Dominio de la Aplicación

Esta capa reúne todos los aspectos del software que tienen que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. Esta capa también recibe el nombre de la capa de la Lógica de la Aplicación.

La capa del Acceso a Datos

Esta capa reúne todos los aspectos del software que tienen que ver con el manejo de los datos persistentes, por lo que también se le denomina la capa de las Bases de Datos. Existen especificaciones de este patrón donde se manipula como será la comunicación entre cada una de las capas definidas.

Arquitectura top-down de capas

Los elementos de una capa i+1 pueden enviar solicitudes de servicio a elementos de la capa inferior i. Típicamente se produce una cascada de solicitudes, es decir para satisfacer una solicitud a una capa i+2, ésta requiere enviar varias solicitudes a la capa i+1; cada una de estas solicitudes a la capa i+1 genera a su vez un conjunto de solicitudes a la capa i y así sucesivamente. Una arquitectura top-down es laxa (o no estricta) si los elementos de una capa i+1 pueden enviar solicitudes de servicio directamente a un elemento de cualquiera de las i capas inferiores.

Arquitectura bottom-up de capas

Cada elemento de una capa i puede notificar a elementos de la capa superior i+1 de que ha ocurrido algún evento de interés (ej. manejadores de dispositivos). La capa i+1 puede juntar varios eventos antes de notificar a su vez al elemento de la capa i+2. Una arquitectura bottom-up también puede ser no estricta si el elemento de la capa i puede notificar a cualquier elemento de cualquier capa superior a la capa i.

Arquitectura bidireccional de capas

En su forma más común involucra dos pilas de N capas que se comunican entre si. El ejemplo más conocido es el de los protocolos en Redes de Computadores.

3) Patrón Cliente- Servidor.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema. La arquitectura cliente-servidor sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico.

Ventajas de la arquitectura cliente-servidor

- Centralización del control: Los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.
- Escalabilidad: Se puede aumentar la capacidad de clientes y servidores por separado.

El servidor de cliente es la arquitectura de red que separa al cliente (a menudo un uso que utiliza un interfaz utilizador gráfico) de un servidor. Cada caso del software del cliente puede enviar peticiones a un servidor. Los tipos específicos de servidores incluyen los servidores de la web, los servidores del uso, los servidores de archivo, los servidores terminales, y los servidores del correo. Mientras que sus propósitos varían algo, la arquitectura básica sigue siendo igual.

4) 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. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

5) Tubería y filtros

Siempre se encuadra este estilo dentro de las llamadas arquitecturas de flujo de datos. Es sin duda alguna el estilo que se definió más temprano y el que puede identificarse topológica, procesual y taxonómicamente con menor ambigüedad. Históricamente el se relaciona con las redes de proceso descriptas por Kahn hacia 1974 y con el proceso secuenciales comunicantes (CSP) ideados por Tony Hoare cuatro años más tarde. Ha prevalecido el nombre de tubería-filtros por más que se sabe muy bien que los llamados filtros no realizan forzosamente tareas de filtrado, como ser eliminación de campos o registros, sino que ejecutan formas variables de transformación, una de las cuales puede ser el filtrado. En uno de los trabajos recientes más completos sobre este estilo, Ernst-Erich Doberkat lo define en estos términos:

“Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Debido a su simplicidad y su facilidad para captar una funcionalidad, es una arquitectura mascota cada vez que se trata de demostrar ideas sobre la formalización del espacio de diseño arquitectónico, igual que el tipo de datos stack lo fue en las especificaciones algebraicas o en los tipos de datos abstractos.”

[Ver **Anexo 1**]. Compilador en tubería-filtro (**Imagen 3**).

6) Estilos Centrados en Datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

7) Arquitecturas de Pizarra o Repositorio

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción se han definidos dos subcategorías principales del estilo:

1. Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
2. Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

[Ver **Anexo 1**]. Arquitectura de Pizarra (**Imagen 4**).

Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados. También se han implementado estilos de este tipo en procesos en lotes de base de datos y ambientes de programación organizados como colecciones de herramientas en torno a un repositorio común. Muchos más sistemas de los que se cree están organizados como repositorios: bibliotecas de componentes reutilizables, grandes bases de datos y motores de búsqueda. Algunas arquitecturas de compiladores que suelen presentarse como representativas del estilo tubería-filtros, se podrían representar mejor como propias del estilo de pizarra, dado que muchos compiladores contemporáneos operan en base a información compartida tal como tablas de símbolos, árboles sintácticos abstractos (AST), etcétera. Así como los estilos lineales de tubería-filtros suelen evolucionar hacia (o ser comprendidos mejor como) estilos de pizarra o repositorio, éstos suelen hacer morphing a estilos de máquinas virtuales o intérpretes.

8) 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. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

9) Arquitecturas Orientadas a Objetos

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización 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, 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. En la semblanza de estos autores curiosamente no se establece como cuestión definitoria el principio de herencia. Ellos piensan que, a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en dicha concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objeto, sino que puede incluir conectores e incluso estilos arquitectónicos enteros.

Si hubiera que resumir las características de las arquitecturas OO, se podría decir que:

- Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.
- Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología (lo mismo que para los componentes en el sentido de CBSE) apenas importa si los objetos son locales o remotos. El mejor ejemplo de OO para sistemas distribuidos es Common Object Request Broker Architecture (CORBA), en la cual las interfaces se definen mediante Interface Description Language (IDL); un Object Request Broker media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.
- En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases.

  En tanto componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

Se puede considerar el estilo como perteneciente a una familia arquitectónica más amplia, que algunos autores llaman Arquitecturas de Llamada-y-Retorno (Call-and-Return). Desde

este punto de vista, sumando las APIs clásicas, los componentes (en el sentido COM y JavaBeans) y los objetos, C-R ha sido el tipo dominante en los últimos 20 años.

10) Estilos 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. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico. Los sistemas basados en reglas, que a veces se agrupan como miembros de la familia de estilos basados en datos, han sido estudiados particularmente por Murrell, Gamble, Stiger y Plant.

11) Arquitectura de Máquinas Virtuales

La arquitectura de máquinas virtuales se ha llamado también intérpretes basados en tablas. De hecho, todo intérprete involucra una máquina virtual implementada en software. Se puede decir que un intérprete incluye un seudo-programa a interpretar y una máquina de interpretación. El seudo-programa a su vez incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución (o registro de activación). La máquina de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución. De este modo, un intérprete posee por lo general cuatro componentes: (1) una máquina de interpretación que lleva a cabo la tarea, (2) una memoria que contiene el seudo-código a interpretar, (3) una representación del estado de control de la máquina de interpretación, y (4) una representación del estado actual del programa que se simula.

El estilo comprende básicamente dos formas o sub-estilos, que se han llamado intérpretes y sistemas basados en reglas. Ambas variedades abarcan, sin duda, un extenso espectro que va desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación, que todo el mundo sabe que implementan un proxy (una especie de nivel de impostura) que encubren al usuario operaciones que en última instancia se resuelven en instrucciones de máquinas afines al paradigma secuencial imperativo de siempre.

[Ver **Anexo 1**]. Arquitectura de Máquinas Virtuales (**Imagen 5**).

El estilo es su conjunto se utiliza habitualmente para construir máquinas virtuales que reducen el vacío que media entre el engine de computación esperado por la semántica del programa y el engine físicamente disponible. Las aplicaciones inscriptas en este estilo

simulan funcionalidades no nativas al hardware y software en que se implementan, o capacidades que exceden a (o que no coinciden con) las capacidades del paradigma de programación que se está implementando.

12) Estilos heterogéneos

Antes de pasar a la familia más fuertemente referida en los últimos tiempos, incluyo en este grupo formas compuestas o indóciles a la clasificación en las categorías habituales. Es por cierto objetable y poco elegante que existan clases residuales de este tipo en una taxonomía, pero ninguna clasificación conocida ha podido resolver este dilema conceptual. En este apartado podrían agregarse formas que aparecen esporádicamente en los censos de estilos, como los sistemas de control de procesos industriales, sistemas de transición de estados, arquitecturas específicas de dominios o estilos derivados de otros estilos, como GenVoca, C2 o REST.

13) Arquitecturas Basadas en Atributos

Aunque algunas otras veces se ha inventado un nuevo estilo para agregarlo al inventario de las variedades existentes, como en este caso, en el de Arch, C2 o REST, la literatura estilística suele ser de carácter reactivo e historicista antes que creativa e innovadora, como si el número de estilos se quisiera mantener deliberadamente bajo. La arquitectura basada en atributos o ABAS fue propuesta por Klein y Klazman. La intención de estos autores es asociar a la definición del estilo arquitectónico un framework de razonamiento (ya sea cuantitativo o cualitativo) basado en modelos de atributos específicos. Su objetivo se funda en la premisa que dicha asociación proporciona las bases para crear una disciplina de diseño arquitectónico, tornando el diseño en un proceso predecible, antes que en una metodología “ad hoc”. Con ello se lograría que la arquitectura de software estuviera más cerca de ser una disciplina de ingeniería, aportando el beneficio esencial de la ingeniería (predictibilidad) al diseño arquitectónico.

El modelo de Klein y Kazman en realidad no tipifica como un estilo en estado puro, sino como una asociación entre la idea de estilo con análisis arquitectónico y atributos de calidad. En este contexto, los estilos arquitectónicos definen las condiciones en que han de ser usados. Además de especificar los habituales componentes y conectores, los estilos basados en atributos incluyen atributos de calidad específicos que declaran el comportamiento de los componentes en interacción. Por ejemplo, en las arquitecturas tubería-filtros, se especifica que se considere de qué manera ha de ser administrada la

performance y se presta atención a los supuestos que rigen el comportamiento de los filtros y al efecto de su re-utilización. Agregando condiciones, un estilo deviene método.

14) Estilos Peer-to-Peer

Esta familia, también llamada de 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. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante broadcast. Miembros de la familia son los estilos basados en eventos, en mensajes (Chiron-2), en servicios y en recursos. Gregory Andrews elaboró la taxonomía más detallada a la fecha de estilos basados en transferencia de mensajes, distinguiendo ocho categorías: (1) Flujo de datos en un sentido a través de redes de filtros. (2) Requerimientos y respuestas entre clientes y servidores. (3) Interacción de ida y vuelta o pulsación entre procesos vecinos. (4) Pruebas y ecos en grafos incompletos. (5) Broadcasts entre procesos en grafos completos. (6) Token passing asincrónico. (7) Coordinación entre procesos de servidor descentralizados. (8) Operadores replicados que comparten una bolsa de tareas.

15) Arquitecturas Basadas en Eventos

Las arquitecturas basadas en eventos se han llamado también de invocación implícita. Otros nombres propuestos para el mismo estilo han sido integración reactiva o difusión (broadcast) selectiva. Por supuesto que existen estrategias de programación basadas en eventos, sobre todo referidas a interfaces de usuario, y hay además eventos en los modelos de objetos y componentes, pero no es a eso a lo que se refiere primariamente el estilo, aunque esa variedad no está del todo excluida. En términos de patrones de diseño, el patrón que corresponde más estrechamente a este estilo es el que se conoce como Observer, un término que se hizo popular en Smalltalk a principios de los ochenta; en el mundo de Java se le conoce como modelo de delegación de eventos.

Desde el punto de vista arquitectónico, los componentes de un estilo de invocación implícita son módulos cuyas interfaces proporcionan tanto una colección de procedimientos (igual que en el estilo de tipos de datos abstractos) como un conjunto de eventos. Los procedimientos se pueden invocar a la manera usual en modelos orientados a objeto, o mediante el sistema de suscripción que se ha descripto.

Los ejemplos de sistemas que utilizan esta arquitectura son numerosos. El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia (bajo la forma de disparadores, por ejemplo), en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos, y en editores sintácticamente orientados para proporcionar verificación semántica incremental.

Las prescripciones de Microsoft para el uso del modelo de eventos señalan que este estilo puede utilizarse cuando:

- Se desea manejar independientemente y de forma aislada diversas implementaciones de una “función” específica.
- Las respuestas de una implementación no afectan la forma en que trabajan otras implementaciones.
- Todas las implementaciones son de escritura solamente o de dispararse-y-olvidar, tal que la salida del proceso de negocios no está definida por ninguna implementación, o es definida sólo por una implementación de negocios específica.

Entre las ventajas enumeradas en relación con el modelo se señalan:

- Se optimiza el mantenimiento haciendo que procesos de negocios que no están relacionados sean independientes.
- Se alienta el desarrollo en paralelo, lo que puede resultar en mejoras de performance.
- Es fácil de empaquetar en una transacción atómica.
- Es agnóstica en lo que respecta a si las implementaciones corren sincrónica o asincrónicamente porque no se espera una respuesta.
- Se puede agregar un componente registrándolo para los eventos del sistema; se pueden reemplazar componentes.

Entre las desventajas:

- El estilo no permite construir respuestas complejas a funciones de negocios.
- Un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea.
- Cuando un componente anuncia un evento, no tiene idea sobre qué otros componentes están interesados en él, ni el orden en que serán invocados, ni el momento en que finalizan lo que tienen que hacer. Pueden surgir problemas de

performance global y de manejo de recursos cuando se comparte un repositorio común para coordinar la interacción.

16) Arquitecturas Orientadas a Servicios

Sólo recientemente estas arquitecturas que los conocedores llaman SOA han recibido tratamiento intensivo en el campo de exploración de los estilos. Al mismo tiempo se percibe una tendencia a promoverlas de un sub-estilo propio de las configuraciones distribuidas que antes eran a un estilo en plenitud. Esta promoción ocurre al compás de las predicciones convergentes de Giga o de Gartner que (después de un par de años de titubeo y consolidación) las visualizan en sus pronósticos y cuadrantes mágicos como la tendencia que habrá de ser dominante en la primera década del nuevo milenio. Ahora bien, este predominio no se funda en la idea de servicios en general, comunicados de cualquier manera, sino que más específicamente va de la mano de la expansión de los Web services basados en XML, en los cuales los formatos de intercambio se basan en XML 1.0 Namespaces y el protocolo de elección es SOAP. SOAP significa un formato de mensajes que es XML, comunicado sobre un transporte que por defecto es HTTP, pero que puede ser también HTTPs, SMTP, FTP, IIOP, MQ o casi cualquier otro, o puede incluir prestaciones sofisticadas de última generación como WS-Routing, WS-Attachment, WS-Referral, etcétera.

Alrededor de los Web services, que dominan el campo de un estilo SOA más amplio que podría incluir otras opciones, se han generado las controversias que usualmente acompañan a toda idea exitosa. Al principio hasta resultaba difícil encontrar una definición aceptable y consensuada que no fuera una fórmula optimista de mercadotecnia. El grupo de tareas de W3C, por ejemplo, demoró un año y medio en ofrecer la primera definición canónica apta para consumo arquitectónico, que no viene mal reproducir aquí:

Un Web service es un sistema de software diseñado para soportar interacción máquina-a-máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el Web service de una manera prescripta por su descripción utilizando mensajes SOAP, típicamente transportados usando HTTP con una serialización en XML en conjunción con otros estándares relacionados a la Web.

En la literatura clásica referida a estilos, las arquitecturas basadas en servicios podrían engranar con lo que Garlan & Shaw definen como el estilo de procesos distribuidos. Otros

autores hablan de Arquitecturas de Componentes Independientes que se comunican a través de mensajes. Según esta perspectiva, no del todo congruente con la masa crítica que han ganado las arquitecturas orientadas a servicios en los últimos años, habría dos variantes del estilo:

- Participantes especificados (named): Estilo de proceso de comunicación. El ejemplar más conocido sería el modelo cliente-servidor. Si el servidor trabaja sincrónicamente, retorna control al cliente junto con los datos; si lo hace asincrónicamente, sólo retorna los datos al cliente, el cual mantiene su propio hilo de control.
- Participantes no especificados (unnamed): Paradigma publish/subscribe, o estilo de eventos.

Los estilos de la familia de la orientación a servicios, empero, se han agrupado de muchas maneras diversas, según surge del apartado que dedicamos a las taxonomías estilísticas. Existen ya unos cuantos textos consagrados a analizar los Web services basados en XML en términos de arquitectura de software en general y de estilos arquitectónicos en particular; especialmente recomendables son el volumen de Ron Schmelzer y otros.

17) Arquitecturas Basadas en Recursos

Una de las más elocuentes presentaciones de arquitecturas Peer-to-Peer ha sido la disertación doctoral de Roy Fielding, elaborada con anterioridad pero expuesta con mayor impacto en el año 2000. Es en ella donde se encuentra la caracterización más detallada del estilo denominado Representational State Transfer o REST. Aunque la literatura especializada tiende a considerar a REST una variante menor de las arquitecturas basadas en servicios, Fielding considera que REST resulta de la composición de varios estilos más básicos, incluyendo repositorio replicado, cache, cliente-servidor, sistema en capas, sistema sin estado, máquina virtual, código a demanda e interfaz uniforme. Fielding no solamente expande más allá de lo habitual y quizá más de lo prudente el catálogo de estilos existentes, sino que su tratamiento estilístico se basa en Perry y Wolf antes que en Garlan y Shaw, debido a que la literatura sobre estilos que se deriva de este último texto sólo considera elementos, conectores y restricciones, sin tomar en consideración los datos, que para el caso de REST al menos constituyen una dimensión esencial.

En síntesis muy apretada, podría decirse que REST 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 de Fielding es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación. De esta manera, el modelado tradicional orientado a objetos deviene innecesario y es reemplazado por el modelado de entidades tales como familias jerárquicas de recursos abstractos con una interfaz común y una semántica definida por el propio HTTP. REST es en parte una reseña de una arquitectura existente y en parte un proyecto para un estilo nuevo. La caracterización de REST constituye una lectura creativa de la lógica dinámica que rige el funcionamiento de la Web (una especie de ingeniería inversa de muy alto nivel), al lado de una propuesta de nuevos rasgos y optimizaciones, o restricciones adicionales: REST, por ejemplo, no permite el paso de cookies y propone la eliminación de MIME debido a su tendencia estructural a la corrupción y a su discrepancia lógica con HTTP. REST se construye expresamente como una articulación compuesta a partir de estilos y sub-estilos preexistentes, con el agregado de restricciones específicas. En la figura 6, RR corresponde a Repositorio Replicado, CS a Cliente-Servidor, LS a sistema en capas, VM a Máquina Virtual y así sucesivamente en función de la nomenclatura referida en la clasificación de Fielding revisada oportunamente. En este sentido, REST constituye un ejemplo de referencia para derivar descripciones de arquitecturas de la vida real en base a un procedimiento de composición estilística, tal como se ilustra en la Figura 6.

[Ver **Anexo 1**]. Composición de REST (**Imagen 6**).

La especificación REST posee algunas peculiaridades emanadas de su lógica orientada a recursos que la hacen diferente de otras implementaciones tales como WebDAV, ebXML, BPML, XLANG, UDDI, WSCK o BPEL4WS que se analizará en un documento específico sobre arquitecturas orientadas a servicios. Sobre una comparación entre dichas variantes puede consultarse el análisis de Mitchell.

2.2 Arquitectura de integración CASE.

2.2.1 Arquitectura del marco de referencia de integración.

Para lograr una buena integración se debe de tener una arquitectura que permita distinguir cada una de las partes del Entorno Integrado, así como se sus funciones. Éste es el Modelo de Arquitectura para el Marco de Referencia de Integración. El contar con tal Marco de Referencia nos permite poder realizar de una manera más sencilla la transferencia de datos

entre las distintas capas de la arquitectura, lo cual simplifica el traspaso entre las distintas herramientas, y por lo tanto, entre las etapas del desarrollo de software.

Capas de la arquitectura de integración:

Para definir la arquitectura de integración de un I-CASE, la dividiremos en distintos niveles de acuerdo a sus funciones, a los que llamaremos capas.

1) Capa de interfaz de usuario.

Esta capa está compuesta por los mecanismos para la comunicación entre el usuario y la máquina. Más específicamente, es el software que compone las interfaces y que permite que se utilicen las herramientas CASE.

2) Capa de herramientas.

Aquí se encuentran en sí las herramientas CASE, las cuales se integran entre sí con la ayuda del resto de la Arquitectura de Integración. Esta capa es la que tiene los servicios que administran y regulan el comportamiento de cada una de las herramientas con la interfaz común.

3) Capa de gestión de objetos

El software de esta capa es el que contiene los servicios de integración en sí. Estos son los módulos que sirven como estándares a las herramientas CASE para conectarse a el depósito de proyectos. También contiene la gestión de la configuración de elementos como control de cambios y versiones.

4) Capa de depósito

Esta capa es la base de datos de los proyectos generados con las herramientas. Esta capa se encarga de almacenar y proveer los datos, así como de su seguridad e integridad, del control del acceso de los usuarios y en general de las funciones que permiten que se comunique la base de datos con las capas superiores [6].

Depósito CASE

Depósito CASE, repositorio, compartido o de proyectos; se le puede llamar de cualquiera de estas formas a la Base de Datos en la cual se almacena la información acerca de un proyecto de desarrollo de software que sea generado por, y utilice, herramientas CASE integradas en un Entorno de Apoyo a Proyectos Integrado. La anterior es una definición que envuelve muchos términos, ya que la creación y especificación de un deposito CASE

compartido cubre muchos conceptos distintos. Una definición distinta seria que “es el mecanismo para definir, almacenar, acceder y administrar toda la información acerca de una empresa, sus datos y sus sistemas de software”. Aquí se define de manera que envuelve no solamente la información de los sistemas, sino de toda una corporación, sus procesos y datos. O podría ser considerarse simplemente que "(el depósito) está reemplazando el limitado concepto anterior del DD... el cual es una lista de los tipos de datos utilizados en el sistema computacional, mientras que el Depósito relaciona toda esta información, añadiéndole significado" A pesar del enfoque que se le de al ámbito que cubrirá el depósito, sus beneficios pueden englobarse en un mismo conjunto:

Beneficios del Depósito CASE:

- Compartir información entre aplicaciones y herramientas.
- Permitir un ambiente multiusuario de herramientas de software.
- Mejorar la comunicación y compartir información entre usuarios.
- Consolidar datos… y eliminar redundancia.
- Incrementar seguridad del sistema.
- Simplificar mantenimiento del sistema.
- Combinar herramientas de distintos proveedores.
- Re-uso de información en distintos etapas del ciclo de desarrollo.
- Simplificar conversiones/migraciones.

Funciones:

El Depósito de un I-CASE sirve no sólo para contener varios tipos de información, sino que también contiene las relaciones entre los diferentes elementos de información, así como las reglas para usar y validar dichos elementos. En de describen las funciones del Depósito de la siguiente manera: "Un Repositorio contribuye a una consistencia mejorada, productividad incrementada y mayor calidad. Provee a los programadores con un mapa consistente de datos y dependencias físicas y lógicas, actuando como un punto único de control que

distribuye la información acerca de los programas y datos a todas las aplicaciones relevantes de desarrollo y producción."

Entre las funciones que cumple el depósito podemos incluir las siguientes

- Integridad de datos, validando los datos que ingresan al Depósito.
- Información compartida, entre las distintas herramientas del I-CASE.
- Integración datos-herramienta, haciendo un modelo de datos para todas las herramientas.
- Integración datos-datos, relacionándolos de manera que se mantenga su integridad y consistencia, permitiendo la correcta funcionalidad del Entorno Integrado.
- Imposición de la metodología, pues se debe de aplicar correctamente para que sea posible almacenar los datos en el Depósito.
- Estandarización de documentos, consecuencia de la definición de los objetos en el Depósito.
- Finalmente el Depósito o Repositorio es el centro y puente de unión de las partes que conforman un I-CASE, otorgando la comunicación confiable y necesaria para poder crear proyectos de desarrollo de software comunicados a través de sus distintas etapas [6].

2.2.2 Módulos principales de una herramienta CASE - I.

1) Repositorio

Base de datos central de la herramienta CASE. El repositorio amplía el concepto de diccionario de datos para incluir toda la información que se va generando a lo largo del ciclo de vida del sistema, como por ejemplo: componentes de análisis y diseño (diagramas de flujo de datos, diagramas entidad-relación, esquemas de bases de datos, diseños de pantallas), estructuras de programas, algoritmos, etc. En algunas referencias se le denomina Diccionario de Recursos de Información. Apoyándose en la existencia del repositorio se efectúan comprobaciones de integridad y consistencia:

- Que no existan datos no definidos.
- Que no existan datos autodefinidos (datos que se emplean en una definición pero que no han sido definidos previamente).

- Que todos los alias (referencias a un mismo dato empleando nombres distintos) sean correctos y estén actualizados.

Las características más importantes del repositorio son:

- Tipo de información. Que contiene alguna metodología concreta, datos, gráficos, procesos, informes, modelos o reglas.
- Tipo de controles. Si incorpora algún módulo de gestión de cambios, de mantenimiento de versiones, de acceso por clave, de redundancia de la información.
- La gestión de cambios y el mantenimiento de versiones. Ayudarán en el caso de que convivan diferentes versiones de la misma aplicación, o se tengan que realizar cambios en la versión en producción y en la de desarrollo, simultáneamente.
- Tipo de actualización. Si los cambios en los elementos de análisis o diseño se ven reflejados en el repositorio en tiempo real o mediante un proceso por lotes (batch). Esto será importante en función a la necesidad de que los cambios sean visibles por todos los usuarios, en el acto.
- Reutilización de módulos para otros diseños. El repositorio es la clave para identificar, localizar y extraer código para su reutilización.

2) Diagramación y modelización.

Diagramas y modelos propuestos y los principales conceptos que estos deben manejar (Incluidos los que debe incorporar ApEM-H):

- Diagrama de clases (Clase, asociación, generalización, dependencia, realización, interfaz.)
- Diagrama de casos de uso (Caso de uso, actor, asociación, extensión, inclusión, generalización.).
- Diagrama de componentes (Componente, interfaz, dependencia, realización.).
- Diagrama de despliegue (Nodo, componente, dependencia, localización.).
- Diagrama de actividad (Estado, actividad, transición de terminación, división, unión.).
- Diagrama de secuencia (Interacción, objeto, mensaje, activación.).
- Diagrama de colaboración (Colaboración, interacción, rol de colaboración, mensaje.).
- Diagrama de estructura de navegación (Clases de navegación, elementos de navegación, composición, asociación, dependencia, uso.).
- Diagrama de Presentación (Clases de presentación, clases Frameset, asociaciones, componentes medias, paquetes.).

Características referentes a los diagramas:

- Número máximo de niveles para poder soportar diseños complejos.
- Número máximo de objetos que se pueden incluir para no encontrarse limitado en el diseño de grandes aplicaciones.
- Número de diagramas distintos en pantalla o al mismo tiempo en diferentes ventanas.
- Dibujos en formato libre con la finalidad de añadir comentarios, dibujos, información adicional para aclarar algún punto concreto del diseño.
- Actualización del repositorio por cambios en los diagramas. Siempre resulta más fácil modificar de forma gráfica un diseño y que los cambios queden reflejados en el repositorio.
- Control sobre el tamaño, fuente y emplazamiento de los textos en el diagrama.
- Comparaciones entre gráficos de distintas versiones. De esta forma será más fácil identificar qué diferencias existen entre las versiones.
- Inclusión de pseudocódigo, que servirá de base a los programadores para completar el desarrollo de la aplicación.
- Posibilidad de deshacer el último cambio, facilitando que un error no conlleve perder el trabajo realizado.

3) Herramienta de prototipado

El objetivo principal de esta herramienta es poder mostrar al usuario, desde los momentos iniciales del diseño, el aspecto que tendrá la aplicación una vez desarrollada. Ello facilitará la aplicación de los cambios que se consideren necesarios, todavía en la fase de diseño. La herramienta será tanto más útil, cuanto más rápidamente permita la construcción del prototipo y por tanto antes, se consiga la implicación del usuario final en el diseño de la aplicación. Asimismo, es importante poder aprovechar como base el prototipo para la construcción del resto de la aplicación.

4) Generador de código

Características más importantes del generador de código:

- Lenguaje generado. Si se trata de un lenguaje estándar o un lenguaje propietario. (Lingo, ActionScript, C#, C++, Object Pascal, PHP.)
- Portabilidad del código generado. Capacidad para poder ejecutarlo en diferentes plataformas físicas y/o lógicas.
- Generación del esqueleto del programa o del programa completo. Si únicamente genera el esqueleto será necesario completar el resto mediante programación.
- Posibilidad de modificación del código generado. Suele ser necesario acceder directamente al código generado para optimizarlo o completarlo.
- Generación del código asociado a las pantallas e informes de la aplicación. Mediante esta característica se obtendrá la interfase de usuario de la aplicación.

5) Módulo generador de documentación.

El módulo generador de la documentación se alimenta del repositorio para transcribir las especificaciones allí contenidas. Algunas características de los generadores de documentación son:

- Generación automática a partir de los datos del repositorio, sin necesidad de un esfuerzo adicional.
- Combinación de información textual y gráfica, lo que hace más fácil su comprensión.
- Generación de referencias cruzadas. Con ello se podrá localizar fácilmente en qué partes de la aplicación se encuentra un determinado objeto o elemento, con el fin de analizar el impacto de un cambio o identificar los módulos afectados por un determinado error.
- Ayuda de tratamiento de textos. Facilidad para la introducción de textos complementarios a la documentación que se genera de forma automática.
- Interface con otras herramientas: procesadores de textos, editores gráficos, etc.

2.3 Conclusiones

En este capítulo se realizó la descripción de las diferentes módulos presentes en una herramienta CASE como es el caso del repositorio, los módulos generadores de informes, los módulos de diagramación y modelación etc., así como se describió cada una de las capas del marco de referencia de integración definido por Pressman conjuntamente con

sus funcionalidades y la interacción entre las diferentes capas de la arquitectura, dirigida la construcción de un entorno CASE integrado OO. De igual manera se describieron los patrones arquitectónicos utilizados en esta investigación resaltando al patrón MVC, de especial significación en la definición la propuesta elaborada en este trabajo la cual será descrita en el siguiente capítulo.

CAPÍTULO 3

PROPUESTA ARQUITECTÓNICA PARA APEM-H

3.1 Introducción.

Es necesario para el entendimiento de este capítulo que se comprenda que la arquitectura de software es:

- Vista estructural de alto nivel: Va a ocurrir en un momento que es previo a la escritura de cualquier clase de código, previo incluso a la elección de un framework de clases o de una herramienta de programación en particular. Ocurre muy tempranamente en todo el proceso del ciclo de vida aunque sigue interviniendo después.
- Define estilo o combinación de estilos para una solución.
- Se concentra en requerimientos no funcionales.

De igual manera cabe señalar que dicho concepto no implica las siguientes ideas que a continuación se señalan y que tienden a confundir:

- Igual en la academia y en la industria: Existe una fuerte discrepancia entre lo que podría ser una adopción universitaria de la arquitectura de software con todos sus elementos de investigación y con todos sus métodos formados y lo que ha sido o lo que es la práctica de la arquitectura de software en la práctica industrial.
- Diseño de software con UML: UML no es reconocida en la academia como una herramienta adecuada para el diseño arquitectónico.
- Naturalmente vinculada a metodología (RUP): Ese vínculo entre RUP y la arquitectura de software recién se esta comenzando a desarrollar, los primeros documentos que formalmente vinculan una cosa con la otra datan de pocos años.
- Naturalmente relacionada con modelado Orientado a Objetos: De hecho existe una especie de contienda entre lo que seria la tendencia que fomenta el desarrollo orientado a objeto y lo que es la arquitectura de software en el sentido académico.

- Las herramientas arquitectónicas generan el código de la aplicación: Este tipo de herramientas recién esta surgiendo en el mercado, herramientas que por lo menos generen la estructura del código de la aplicación.

3.2 Fases del ciclo de vida que debe cubrir ApEM-H:

En este sentido ApEM-H presenta una característica peculiar debido que constituye un requisito de estricto cumplimiento que la herramienta pueda abarcar dos importantes fases del ciclo de vida del desarrollo de sistemas para el software educativo en el contexto productivo cubano y sus particularidades, dichas fases son:

1. Análisis y diseño.
2. Implementación (generación de código fuente básico).

Como se puede apreciar ambas fases pertenecen a distintos niveles dentro del ciclo de desarrollo del software. Sin embargo en ApEM-H se integrarán ambas fases, constituyendo así una herramienta CASE integrada que no llega a abarcar todo este ciclo, pero que si cubre las que son consideradas sus etapas fundamentales. En el siguiente esquema se explica de forma grafica lo aquí argumentado:

[Ver **Anexo 1**]. Fases que cubre ApEM-H (**Imagen 7**).

[Ver **Anexo 2**]**.** Comparación de las herramientas CASE de acuerdo a la fases del ciclo de vida que soportan (**Tabla 2**).

Ventajas de los Entornos Integrados.

- Transferencia de Información: Se pueden transferir la información entre una “sub-herramienta” y otra. De esa manera algunos de los artefactos, o parte de estos artefactos construidos en una “sub-herramienta”, podrán ser automáticamente generados en otras.
- Reducción de Esfuerzos: Al integrar todas estas “sub-herramientas” y posibilitar así la automatización de muchos procesos, obviamente se reduce el tiempo de desarrollo del proyecto tanto como el esfuerzo que se requiere para finalizar el mismo.
- Aumento del Control del proyecto: También es fácil de deducir que por tanto el control que se lleva sobre el proyecto será mayor, puesto que se tiene conocimiento de varias versiones dentro del mismo y así se podrá tener la opción de deshacer cambios o rehacer los mismos, en dependencia de la necesidad del usuario.

- Mejor coordinación: Todo esto implica una mayor coordinación entre las distintas fases del proyecto, puesto que la herramienta se encarga de hacer operaciones que son transparentes al usuario y por tanto tener un margen mucho menor de error.

3.3 Patrones arquitectónicos que debe soportar para la modelación de ApEM-L.

A continuación se explican las características principales de los patrones arquitectónicos presentes en ApEM-L, que se van a modelar en ApEM-H, los mimos son una derivación del patrón Modelo-Vista-Controlador, explicado en el Capítulo 2 del presente trabajo:

El patrón Modelo – Vista – Controlador Multi-Media (MVCMM).

En la variante para modelar aplicaciones multimedia (MVCMM) se diversifica la clase modelo incorporando al esquema dos nuevos tipos de clases: la modelo dinámica y la modelo estática, la cual a su vez se subdivide en las clases media y lógica de la aplicación. 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 que el usuario donde se refleja los cambios en el modelo. 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.

La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente. El flujo recién descrito en el párrafo anterior deja claramente las siguientes responsabilidades a las clases conformantes de la arquitectura:

Clase Vista: Recibe las peticiones del usuario al sistema. Muestra la información de salida al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

Clase Modelo: Encapsula los datos y las funcionalidades. Gestiona el procesamiento y almacenamiento de la información. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.

Clase Controladora: Reciben las entradas para la gestión de las mismas, a partir de las clases vistas, 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. Gestionan el modelo que realizará el procesamiento así como la vista que generará la respuesta al usuario.

El patrón Modelo – Vista – Controlador – Entidad (MVC-E)

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. (Pressman, 2002)

Este enfoque unido a las características distintivas del software educativo producido en Cuba y descritas en la introducción de este trabajo, ha traído como consecuencia el análisis de una variante de solución al MVCMM para las aplicaciones educativas cubanas, donde se descarguen las responsabilidades de la clase modelo concernientes al procesamiento y almacenamiento de la información persistente de las aplicaciones, incorporando una nueva clase al modelo denominada Modelo Entidad, con dos tipos fundamentales, la clase

Modelo-Entidad-Media y Modelo-Entidad- Persistente. La primera de estas con la responsabilidad de agrupar las clases que identifican las medias y su árbol de jerarquía en la aplicación y la segunda tiene como responsabilidad la gestión de la información persistente, que antes sobrecargaba a la clase Modelo del patrón MVC original. Esta variación a su vez sustenta las características actuales de los sistemas multimedia como son: comunicación con bases de datos, archivos XML, o sistemas externos.

3.4 Arquitectura de capas definida para ApEM-H.

La herramienta CASE ApEM-H está compuesta por las herramientas ApEM-U-CASE y ApEM-L-CASE, cuyas arquitecturas van a ser descritas posteriormente.

La arquitectura que se propone en este trabajo investigativo está basada en el modelo arquitectónico definido por para ambientes CASE integrados conocido como modelo de referencia de integración.

[Ver **Anexo 1**]**.** Modelo de integración de cuatro capas (**Imagen 8**).

Este modelo de integración permite distinguir cada una de las partes del ambiente CASE integrado ApEM-H (Herramienta para la Modelación de Aplicaciones Educativas), así como sus funcionalidades. Otro beneficio importante es que facilita la transferencia de información entre los usuarios y las herramientas involucradas que en este caso son las herramientas ApEM-U-CASE y ApEM-L-CASE en el proyecto de desarrollo, y por tanto en el entre las etapas del proceso. Además de las ventajas y beneficios que ofrece la arquitectura por capas en sí para conceptualización lógica y funcional de la arquitectura de ApEM-H, como la reusabilidad, véase que si el día de mañana queremos cambiar, por ejemplo, de repositorio de datos, seguramente el impacto se concentra en la capa de depósito de datos pero no en las otras, además de la división en capas es una división lógica a la hora de codificar las clases de la aplicación, para abaratar sus costos de desarrollo y su posterior mantenimiento.

Se puede inferir que el modelo definido por Pressmnan es modelo basado en la arquitectura por capas, además se ha elegido este modelo arquitectónico de Pressman también porque el mismo esta dirigido a la construcción de ambientes CASE integrados orientados a Objetos (I-CASE OO), elemento determinante a la hora de la adopción de dicho modelo arquitectónico para la conceptualización de la arquitectura ApEM-H, dígase que la ApEM-H está concebida para que su implementación y construcción, así como el lenguaje que soporta (ApEM-L) sea bajo el enfoque o paradigma orientado a objeto(OO)

1. Capa de Interfaz:

En esta capa se implementa el software o módulo para las administración de la interfaz grafica de usuario, dándose entonces la situación de que en esta capa cuente con componentes gráficos compuesto o elementales, la interfaz de usuario será independiente de los servicios u funcionalidades ofrecidas por ApEM-H, proporcionando el mismo aspecto para todas las herramientas, así como los mecanismos para acceder a las herramientas involucradas dentro de la herramienta CASE integrada ApEM-H, lo que implica se debe proveer la interfaz de acceso a todas las funcionalidades de la herramienta a desarrollar, lo que implica la definición de las convenciones del diseño de pantalla, nombres y organización del menú así como los iconos a utilizar, es decir los mecanismos de comunicación entre el usuario y la aplicación para acceder a los servicios de la que brindará la herramienta, los cuales son posibles mediante el uso del patrón Modelo-Vista-Controlador(MVC) , esta capa se usa para proveer el protocolo de presentación y para la presentación visual de todos los elementos presentes en la capa Herramienta así como la edición de modelos y , que a continuación será descrita y por supuesto para la interacción con las funcionalidades del OOCE. Aquí se implementa la biblioteca de objetos de visualización.

2. Capa de Herramientas:

Esta capa proporciona la funcionalidad que garantiza el desarrollo y consistencia de los diferentes modelos que se desarrollan durante el proceso de elaboración del software, en sus diferentes etapas y según las necesidades de la organización. El patrón MVC permite la interacción de la capa de interfaz con esta capa de herramientas, de manera que si el usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa una tecla que indique la automatización de alguna operación del sistema por ejemplo a partir del diagrama de secuencia realizar el de colaboración con la tecla F5), el controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario, es decir usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas como la anterior, etc. Los eventos son traducidos a solicitudes de servicio (“service requests”) para el modelo. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback, El controlador accede o gestiona al modelo correspondiente que es el encargado de cumplir la función encomendada por el componente controlador, este modelo va a ser la herramienta CASE (ApEM-U-CASE o (ApEM-L-CASE) que brinda el servicio asociado a este evento y que esta lo ejecute, además el controlador también gestiona la vista que generará la respuesta al usuario.

Aquí se encuentran las herramientas CASE que integran el ambiente CASE integrado ApEM-H, y por supuesto esta capa se encarga de los servicios de gestión de herramientas (SGH).Más específicamente, controla el comportamiento y las funciones de las herramientas dentro del entorno. Esta capa es la que tiene los servicios que administran y regulan el comportamiento de cada una de las herramientas con la interfaz común esto es posible con el uso del patrón MVC, haciendo transparente a la capa de interfaz las funciones ejecutadas por las herramientas y de esta manera a los ojos del usuario. También de provee los mecanismos sincronización y comunicación multitarea, coordinación y flujo información, seguridad y auditoria y recolección de métricas.

3. Capa de Gestión de Objetos:

Es la que contiene los servicios de integración en sí, el mismo provee módulos que sirven como estándares a las herramientas CASE para conectarse a el depósito de proyectos o repositorio, por lo que proporciona la integración de las herramientas. También contiene la gestión de la configuración de elementos como el control de cambios y versiones, llevando a cabo dichas funciones, además garantiza la consistencia entre los modelos de análisis, diseño e implementación garantizando así la difusión de los cambios ocurridos entre las herramientas responsables de estas funcionalidades, esta capa permite que los cambios realizados por una de las herramientas en un ámbito sean concurrentemente reflejados en otro ámbito que depende del anterior, es decir permitir la gestión de enlaces, de manera que sea capaz de identificar y evaluar los efectos de un cambio que relacione a todas las herramientas del modelo. Véase que aquí se tiene también otro componente o clase que la que gestiona todos estos procedimientos que realiza el primer componente anteriormente descrito y que es la que interacciona con la capa de herramientas directamente, por tanto esta clase oculta a la capa de herramientas los módulos que realizan los funcionalidades anteriores como la gestión de enlaces entre herramientas y la consistencia entre el modelo análisis y diseño e implementación, además de los mecanismos para conectarse el repositorio, etc. En esta capa también se generan los meta datos se construye y meta modelo.

La herramienta ApEM-H debe generar automáticamente información acerca de los datos con los que trabaja, es decir, datos sobre sus propios datos, esto esta generación es denominada meta datos. Los datos que genera pueden ser:

- Definiciones de objetos (de que tipo son, que representación gráfica tienen, con que otros objetos se relacionan).

- Relaciones y dependencias entre objetos de distinto nivel lógico. Por Ej.: un proceso en un DFD (Diagrama de Flujo de Datos).
- Reglas de diseño aplicadas a su propio software, por Ej.: las distintas formas válidas de dibujar y balancear un DFD.
- Procedimientos (fases estándar, hitos, informes, etc.) y sucesos (revisiones, finalizaciones, informes de problemas, peticiones de cambios, etc.) del flujo de trabajo (proceso).

ApEM-H debe constar con un meta modelo, el cual debe ser un modelo de modelos, es un modelo que define los pasos a seguir para construir otro modelo; en este caso modelos de sistemas de información. El mismo ha de ser el patrón que rija la generación y almacenamiento de la información del proyecto.

4. Capa de Depósito de Datos:

En esta capa se implementa el repositorio de la herramienta ApEM-H, el mismo es una base de datos de los proyectos generados con las herramientas ApEM-U-CASE y ApEM-L-CASE, el mismo se encarga almacenar y proveer datos, de la seguridad e integridad de los mismos, así como del control de acceso de usuarios y en general de las funciones que permiten que se comunique la base de datos con las capas superiores, todo lo anteriormente descrito implica almacenar el problema a resolver, información sobre el ámbito del problema, los modelos que solucionan el problema, las reglas e instrucciones relacionadas con la metodología, la información concerniente a la gestión del proyecto(recursos, presupuesto, calendario, etc.).Para proporcionar un menor acoplamiento se implementa en esta capa un componente que sea el que permite la interacción con la capa de gestión objetos, el cual gestiona los servicios o funcionalidades que presta el repositorio.

El repositorio CASE de ApEM-H también ha de seguir los principios y características descritos a continuación:

Este repositorio es la base de de datos de los proyectos generados con las herramienta, se encarga de almacenar y proveer los datos, así como de su seguridad e integridad, del control del acceso de los usuarios.

El depósito CASE o repositorio compartido de proyectos; se le puede llamar de cualquiera de estas formas a lugar donde en el cual se almacena la información acerca de un proyecto de desarrollo de software que sea generado por, y utilice, herramientas CASE integradas en

un Entorno de Apoyo a Proyectos Integrado. La anterior es una definición que envuelve muchos términos, ya que la creación y especificación de un deposito CASE compartido cubre muchos conceptos distintos, “es el mecanismo para definir, almacenar, acceder y administrar toda la información acerca de una empresa, sus datos y sus sistemas de software. Aquí se define de manera que envuelve no solamente la información de los sistemas, sino de toda una corporación, sus procesos y datos. O podría ser considerarse simplemente que "(el depósito) está reemplazando el limitado concepto anterior del DD... el cual es una lista de los tipos de datos utilizados en el sistema computacional, mientras que el Depósito relaciona toda esta información, añadiéndole significado".

A pesar del enfoque que se le de al ámbito que cubrirá el depósito, sus servicios pueden englobarse en un mismo conjunto:

Beneficios del Depósito CASE:
• Compartir información entre aplicaciones y herramientas
• Mejorar la comunicación y compartir información entre usuarios
• Consolidar datos… y eliminar redundancia
• Incrementar seguridad del sistema
• Simplificar mantenimiento del sistema
• Combinar herramientas de distintos proveedores
• Re-uso de información en distintos etapas del ciclo de desarrollo
• Simplificar conversiones/migraciones

Funciones:

El Depósito de un I-CASE sirve no sólo para contener varios tipos de información, sino que también contiene las relaciones entre los diferentes elementos de información, así como las reglas para usar y validar dichos elementos. En de describen las funciones del Depósito de la siguiente manera:
"Un Repositorio contribuye a una consistencia mejorada, productividad incrementada y mayor calidad. Provee a los programadores con un mapa consistente de datos y dependencias físicas y lógicas, actuando como un punto único de control que distribuye la información acerca de los programas y datos a todas las aplicaciones relevantes de desarrollo y producción."

Entre las funciones que cumple el depósito podemos incluir las siguientes:

- Integridad de datos, validando los datos que ingresan al Depósito.
- Información compartida, entre las distintas herramientas del I-CASE.
- Integración datos-herramienta, haciendo un modelo de datos para todas las herramientas.
- Integración datos-datos, relacionándolos de manera que se mantenga su integridad y consistencia, permitiendo la correcta funcionalidad del Entorno Integrado.
- Imposición de la metodología, pues se debe de aplicar correctamente para que sea posible almacenar los datos en el Depósito.
- Estandarización de documentos, consecuencia de la definición de los objetos en el

Depósito:

Finalmente el Depósito o Repositorio es el centro y puente de unión de las partes que conforman un I-CASE, otorgando la comunicación confiable y necesaria para poder crear proyectos de desarrollo de software comunicados a través de sus distintas etapas.

En esta arquitectura se evidencia lo siguiente:

- Cohesión: En la arquitectura propuesta se diferencia claramente la descomposición de funciones.
- Acoplamiento: Dado por el uso del patrón MVC en y por los componentes en cada una de las capas que hacen transparentes a sus respectivas capas superiores las funcionalidades desarrolladas por la capa en la que ellos se encuentran. Si se cambia, por ejemplo, el medio de comunicación o de almacenamiento de una de las partes, la otra, que por ejemplo hace la presentación, no tiene porque enterarse, y viceversa; por lo que permite un bajo acoplamiento.
- Adaptabilidad: Como una consecuencia del bajo acoplamiento y de la alta cohesión se consigue alta adaptabilidad.

3.4.1 Arquitectura de la herramienta ApEM-U-CASE

Herramienta CASE Superior para la Modelación de Aplicaciones Educativas. Orientada a objeto que compone la Herramienta CASE ApEM-H, esta herramienta constará con una arquitectura de tres capas, siguiendo el modelo propuesto por (Losavio et al., 1999) para un entorno CASE orientado al objeto (OOCASE). Dicho modelo propone las siguientes tres capas: interfaz, semántica e integración.

1. Capa Interfaz:

Para la capa interfaz de la Herramienta se debe tener en cuenta que la interfaz de usuario deber ser independiente de la funcionalidad ofrecida por la herramienta. Con este fin se ha utilizado como base del diseño de la interfaz de usuario el patrón Modelo-Vista-Controlador (MVC), ampliamente detallado en (Buschmann et al., 1996).

En esta capa se debe proveer la interfaz de acceso a todas las funcionalidades de la herramienta a desarrollar, esto a través del patrón MVC que además va a permitir la interacción con la capa semántica y a su vez el componente Controlador decide cual será el servicio a ejecutar por el componente modelo ambos componentes han de estar en la capa semántica, a diferencia de la vista que se encuentra en la capa interfaz y que es la que genera el evento que es recibido por el controlador. En esta capa se hace la definición de las convenciones del diseño de pantalla, nombres y organización del menú así como los iconos a utilizar, es decir los mecanismos de comunicación entre el usuario y la aplicación para acceder a los servicios de la que brindará la herramienta, esta capa se usa para proveer el protocolo de presentación y para la presentación visual de todos los elementos presentes en la capa semántica así como la edición de modelos y, que a continuación será descrita y por supuesto para la interacción con las funcionalidades del OOCE. Se compone del módulo o software para la administración de la interfaz grafica del usuario.

2. Capa Semántica:

En esta capa se deben implementar los servicios que brindará la herramienta, en el caso de este ambiente U-CASE OO, se centrará en los servicios orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: planificación, análisis y diseño siguiendo las normas establecidas por el estándar ApEM-L, los que se pueden desglosar en definición de requisitos del sistema y sus propiedades que permiten crear y modificar diagramas E/R,(Entidad-Relación), diagramas de flujo de datos, diagramas de estructura de cuadros, diagramas de clases, etc., para ello se implementan módulos de diagramación y modelación, que son los que cumplen con estas responsabilidades. También son importantes servicios dirigidos al diseño de pantallas, generación de menús, generación de informes y lenguajes de especificación ejecutables, estos deben describir los aspectos fundamentales de un sistema, obteniéndose beneficios considerables en la documentación gráfica y en la integración de funciones y relaciones, de manera que brinde un grado de la capacidad de análisis y verificación de especificaciones que soporta la herramienta, no solo sintáctica sino también semántica, como, por ejemplo, la capacidad de normalizar un diagrama de datos. Todo esto por supuesto siguiendo las

normas, principios y características establecidas por el Lenguaje para la Modelación de Software Educativo (ApEM-L) que soporta ApEM-U-CASE.

La herramienta debe tener un modulo de comprobación de errores, el cual proveerá un conjunto de facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta. El cual provee analizador de sintaxis que verifique el correcto uso de la información introducida en la herramienta de acuerdo por el lenguaje ApEM-L soportado por ApEM-U-CASE, este componente es muy importante para ayudar a desarrollar o no cometer errores de sintaxis al momento de crear los diagramas de clases y luego con la herramienta ApEM-L-CASE generar un código más completo y preciso, servicio este muy deseable a para la modelación de las aplicaciones educativas con el lenguaje ApEM-L y que reunirá ambas funciones la herramienta ApEM-H, que es la integrada que reúne a ambas herramientas. En esta capa también se generan los meta datos se construye y meta modelo.

La herramienta ApEM-H debe generar automáticamente información acerca de los datos con los que trabaja, es decir, datos sobre sus propios datos, esto esta generación es denominada meta datos. Los datos pueden ser:

- Definiciones de objetos (de que tipo son, que representación gráfica tienen, con que otros objetos se relacionan).
- Relaciones y dependencias entre objetos de distinto nivel lógico. Por Ej.: un proceso en un DFD (Diagrama de Flujo de Datos).
- Reglas de diseño aplicadas a su propio software, por Ej.: las distintas formas validas de dibujar y balancear un DFD.

ApEM-U-CASE debe constar con un meta modelo, el cual debe ser de modelos, es un modelo que define los pasos a seguir para construir otro modelo; en este caso modelos de sistemas de información. El mismo ha de ser el patrón que rija la generación y almacenamiento de la información del proyecto.

3. Capa Integración:

Es donde se debe implementar el Repositorio CASE de ApEM-UCASE, este repositorio debe almacenar el problema a resolver, información sobre el ámbito del problema, los modelos que solucionan el problema, las reglas e instrucciones relacionadas con la metodología, la información concerniente a la gestión del proyecto (recursos, presupuesto, calendario, etc.). Aquí se tiene un componente que es el que posibilita el intercambio entre

la capa semántica y las funcionalidades que despliega el repositorio, haciendo transparente a la capa semántica como es que el repositorio realiza sus funciones, de manera que este componente gestiona las funciones que presta este depósito de datos, y permite una gran flexibilidad a la arquitectura y una gran reusabilidad.

Servicios que debe prestar el repositorio:

Los servicios que debe proporcionar el repositorio CASE, de ApEM-U-CASE, son:

- Independencia de los datos
- Almacenamiento no redundante.
- Independencia de los datos.
- Acceso de alto nivel: un único mecanismo de acceso para todas las herramientas.
- Independencia de los datos.
- Control de transacciones.
- Seguridad.
- Consulta de los datos y gestión de informes.
- Apertura: mecanismo sencillo para importar/exportar datos.
- El modelo de datos debe permitir establecer una gran diversidad de relaciones entre los datos que los componen.
- Información compartida.
- Seguimiento de la metodología.

3.4.2 Arquitectura de la herramienta ApEM-L-CASE

Herramienta Inferior para La Modelación de Aplicaciones Educativas. De igual manera la misma se constará también con una arquitectura de de 3 capas: interfaz, semántica e integración, como su sucesora ApEM-U-CASE, anteriormente descrita, y es orientada a objeto también pues modelará y será construida bajo el paradigma OO.

De igual manera para la realización de la capa interfaz se ha determinado el patrón arquitectónico MVC, así como las demás funcionalidades a describir en esta capa serán exactamente iguales a las de ApEM-U-CASE. Lo mismo ocurre con capa de integración, las funcionalidades serán prácticamente las mismas, por supuesto con el correspondiente sentido lógico asociado al contesto de cada herramienta respectivamente.

La diferencia notable se encuentra en la capa semántica donde a su vez se implementarán los servicios que debe brindar la ApEM-L-CASE, los cuales serán dirigidos a las últimas fases del desarrollo, estos servicios son la generación de código, como por ejemplo la generación de la estructura estática de un sistema de software, es decir la arquitectura de la estructura de las clases participantes con sus atributos y los cuerpos vacíos de sus operaciones, esto a partir de diagramas de clases definidos utilizando ApEM-L.

Por ultimo de similar modo que la herramienta ApEM-U-CASE en esta capa la semántica también implementará un módulo de comprobación de errores, el cual deber proporcionar un conjunto de elementos que permitan llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta. Y también aquí se generan los meta datos y crean los meta modelos.

3.5 Herramienta propuesta para el desarrollo de la aplicación ApEM-H.

Eclipse.

Resumen.

5. Desarrollador: Eclipse Foundation.
6. Última versión: 21 de Febrero de 2008.
7. SO: Multiplataforma.
8. Género: IDE, Java SDK, C/C++
9. Sitio Web: http://www.eclipse.org/

Eclipse es un entorno de desarrollo integrado de código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo

Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Arquitectura.

La base para Eclipse es la Plataforma de cliente enriquecido (del Inglés Rich Client Platform RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal - inicio de Eclipse, ejecución de plugins
- OSGi - una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace - manejo de archivos, manejo de texto, editores de texto
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes

Los widgets de Eclipse están implementados por una herramienta de widget para Java llamada SWT, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window Toolkit (AWT) o Swing. La interfaz de usuario de Eclipse también tiene una capa GUI intermedia llamada JFace, la cual simplifica la construcción de aplicaciones basada en SWT.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el

ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".

En cuanto a las aplicaciones clientes, eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto wysiwyg hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de meta data en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

El 28 de junio de 2005 fue liberada la versión 3.1 que incluye mejoras en el rendimiento, el soporte de Java 5.0, mejor integración con Ant (incluido debugger) y CVS.

La última versión estable es la 3.3 y fue liberada el 25 de junio de 2007. Dentro de la rama 3.3, su versión actual más avanzada es la 3.3.1.1, liberada el 23 de octubre de 2007.

Características.

La versión actual de Eclipse dispone de las siguientes características:

1. Editor de texto.
2. Resaltado de sintaxis.
3. Compilación en tiempo real.
4. Pruebas unitarias con JUnit.
5. Control de versiones con CVS.

6. Integración con Ant.
7. Asistentes (wizards): para creación de proyectos, clases, tests, etc.
8. Refactorización.
9. Asimismo, a través de "plugins" libremente disponibles es posible añadir:
   - Control de versiones con Subversion.
   - Integración con Hibernate.

Licencias

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia Pública General de GNU (GNU GPL) Mike Milinkovich, de la fundación Eclipse comentó que el cambio a la GPL será considerado cuando la versión 3 de la GPL sea liberada

Idiomas

En julio de 2006, los siguientes paquetes de lenguajes están disponibles para Eclipse 3.2.x (En orden alfabético): Alemán, Árabe, Checo, Chino Simplificado, Chino Tradicional, Coreano, Español, Francés, Húngaro, Inglés, Italiano, Japonés, Polaco, Portugués (Brasil) y Ruso.

3.6 Validación de la propuesta.

Para comprobar que la arquitectura propuesta en este trabajo investigativo para la construcción de la herramienta para la modelación de aplicaciones educativas (ApEM-H) se definió correctamente, es necesario validar dicha propuesta.

Para validar técnicamente la propuesta se puede utilizar el método de experto, que permite tomar decisiones para aceptar o no le propuesta de acuerdo con los criterios definidos.

Para llevar a cabo el desarrollo del mismo se efectuaron un conjunto de pasos, descritos a continuación.

Se elaboran los criterios de evaluación de acuerdo a las características de la propuesta y se organizan por grupos. Se le asigna un peso cada grupo de criterios de acuerdo al por ciento que representa cada grupo del total y los intereses a evaluar.

Se les entrega a los expertos la propuesta para que estudien el tema a evaluar y dos modelos, uno para que valore el peso relativo de cada criterio.

Grupo No.1……50

Grupo No.2……....25

Grupo No.3…….25

Ahora se determina que criterios abarcó cada grupo

Grupo No.1:

- Representación de las funcionalidades que debe realizar la capa de interfaz.
- Representación de las funcionalidades que debe realizar la capa de herramientas.
- Representación de las funcionalidades que debe realizar la capa de gestión de objetos.
- Representación de las funcionalidades que debe realizar la capa de depósito de datos.

Grupo No 2:

- Definición de la arquitectura de la herramienta ApEM-U-CASE.
- Definición arquitectura de la herramienta ApEM-L-CASE.

Grupo No 3:

- Representación de las características de un OOCASE en especial su arquitectura.
- Aplicabilidad de la arquitectura para la construcción de ApEM-H.

Se organiza un comité de expertos con una cantidad mínima de 7 teniendo en cuenta su especialidad, grado científico y currículo y se les entrega a los expertos la propuesta para que estudien el tema a evaluar y dos modelos, uno para que valore el peso relativo de cada criterio y y otro para realizar una evaluación cuantitativa de cada criterio con una escala de 1-5 y la apreciación cualitativa con una clasificación final del proyecto en excelente, bueno, aceptable, cuestionable y malo.

Después de recibir los valores del peso de cada criterio se construye la siguiente tabla

[Ver **Anexo 2**]**.** Resultado del trabajo de expertos (**Tabla 3**).

Se verificó la consistencia en el trabajo de los expertos, para lo que se utiliza el coeficiente de concordancia de Kendall y el estadígrafo Chi cuadrado (X^2). Se sigue el procedimiento siguiente:

- Sea C el número de criterios que van a evaluarse y E el número de expertos que realizan la evaluación.
- Para cada criterio se determina:

  ∑E: Sumatoria del peso dado por cada experto

  Ep: Puntuación promedio del peso dado por cada experto

  M∑E: media de los ∑E

  ΔC: Diferencia entre ∑E y M∑E

- Se determina la desviación de la media, que posteriormente se eleva al cuadrado para obtener la dispersión (S) por la expresión

  $S = \sum (\sum E - \sum \sum E / C)^2$

  $S = 78$

- Conociendo la dispersión se puede calcular el coeficiente de concordancia de Kendall (W)

  $W = S / (E^2 (C^3 – C)/ 12)$

  W=78 /2058

  W=0.038

- El coeficiente de concordancia de Kendall permite calcular el Chi cuadrado real

  $X^2 = E (C-1) W$

  $X^2 = 1.862$

Los valores obtenidos se muestran en la siguiente tabla

[**Ver Anexo 2**]. Cálculo de concordancia de Kendall (**Tabla 4**).

- El Chi cuadrado calculado se compara con el obtenido del las tablas estadísticas para un α (coeficiente de confianza) de 0.9.

  Se tiene que:

  $X^2 (\alpha, c-1) = X^2 (0.9, 8-1) = X^2 (0.9, 7) = 2.8331$

X^2real < X^2 (α, c-1)

X^2 real =1.862

1.862< 2.8331

Existe concordancia en el trabajo de expertos

Después de comprobar la consistencia del trabajo de expertos se puede definir el peso relativo de cada criterio (P). Conociendo el peso de cada criterio y la calificación dada por los evaluadores en una escala de 1-5 se puede construir la tabla de calificación de cada criterio, para obtener el valor de de P × c, donde (c), es el criterio promedio concebido por los expertos.

[Ver **Anexo 5**]**.** Calificación de cada criterio (**Tabla 5**).

Se calcula el Índice de aceptación del proyecto (IA)

IA = ∑ (P × c) / 5

IA = 0.88

Rangos predefinidos de Índice de Aceptación.

IA > 0,7 Existe alta probabilidad de éxito

0,7 > IA > 0,5 Existe probabilidad media de éxito

0,5 > IA > 0,3 Probabilidad de éxito baja

0,3 > IA Fracaso seguro

Por lo que la probabilidad de éxito es alta IA > 0.7

La validación mediante el juicio de expertos arroja un resultado positivo para la propuesta de arquitectura, según se puede apreciar en la tabla anterior, de manera general la representación de las funcionalidades que debe realizar la capa de interfaz, herramientas, gestión de datos y depósito de datos son aceptados. Lo mismo ocurre con definición de la arquitectura de la herramienta ApEM-U-CASE y definición de la arquitectura de la herramienta ApEM-L-CASE, la representación de las características de un OOCASE en

especial su arquitectura y de la aplicabilidad de la arquitectura para la construcción de ApEM-H. Esta validación da una garantía de que la utilización de la arquitectura definida en esta investigación tendrá éxito a la hora de construir la herramienta ApEM-H.

3.7 Conclusiones

La validación realizada en este capítulo es la garantía de la arquitectura propuesta a lo largo de este trabajo fue definida correctamente. El juicio de los expertos que estudiaron un resumen del trabajo brinda la total confianza y asegura la obtención de un resultado positivo a todo aquel equipo que aplique la propuesta arquitectónica en el desarrollo y construcción de la Herramienta para la Modelación de Aplicaciones Educativas (ApEM-H), pues a través del coeficiente de concordancia de Kendall utilizando la tabla de valores y el estadígrafo Chi cuadrado (X2), se determinó la que el trabajo de los expertos era consistente. Además se realizo la descripción en este capítulo de la propuesta de arquitectura de esta investigación en la cual se utilizó el modelo arquitectónico definido por Pressman en el año 2001 para una entorno o ambiente CASE integrado OO, para la descripción de ApEM-H y el modelo arquitectónico definido por Losavio en el año 1998 para la construcción de un ambiente CASE OO para la descripción de la arquitectura de las herramientas ApEM-U-CASE y ApEM-L-CASE, que componen a ApEM-H, así como el patrón arquitectónico que se utilizó en cada una de las mismas que fue el MVC.

CONCLUSIONES

Como resultado de este trabajo se realizó una investigación que permitió realizar una correcta definición del modelo arquitectónico que ha de tener la Herramienta para la Modelación de Aplicaciones Educativas ApEM-H, con sus diferentes subsistemas o módulos, así como las funcionalidades estos deben desarrollar y en las de acuerdo a las capas en las que se implementan, también fue importante para la descripción de la propuesta arquitectónica definida en esta investigación el uso del patrón Modelo Vista Controlador, el cual permite la interacción entre diferentes capas de la arquitectura propuesta en este trabajo, además de independizar a la capa interfaz de usuario presente en el modelo arquitectónico propuesto para construir a ApEM-H, de las funcionalidades que desempeñan las herramientas CASE que se cuya arquitectura se describe en el capítulo 3 específicamente. Además se tiene que en este trabajo se describe y justifica de manera consecuente el uso de del modelo de Pressman para la construcción de ambientes o entornos CASE integrados orientados a objetos, también conocido como marco de referencia de integración y de modelo definido por Losavio para la contracción de ambientes CASE orientados a objetos. Por último se realizó la validación mediante el juicio de expertos arroja un resultado positivo para la propuesta de arquitectura de manera general la representación de las funcionalidades que debe realizar la capa de interfaz, herramientas, gestión de datos y depósito de datos son aceptados. Lo mismo ocurre con definición de la arquitectura de la herramienta ApEM-U-CASE y definición de la arquitectura de la herramienta ApEM-L-CASE, la representación de las características de un OOCASE en especial su arquitectura y de la aplicabilidad de la arquitectura para la construcción de ApEM-H. Esta validación da una garantía de que la utilización de la arquitectura definida en esta investigación tendrá éxito a la hora de construir la herramienta ApEM-H. El juicio de los expertos que estudiaron un resumen del trabajo brinda la total confianza y asegura la obtención de un resultado positivo a todo aquel equipo que aplique la propuesta arquitectónica en el desarrollo y construcción de la Herramienta para la Modelación de Aplicaciones Educativas (ApEM-H), pues a través del coeficiente de concordancia de Kendall utilizando la tabla de valores y el estadígrafo Chi cuadrado (X2), se determinó la que el trabajo de los expertos era consistente.

RECOMENDACIONES

Que se utilice el modelo arquitectónico propuesto en este trabajo investigativo para la construcción de la Herramienta para la Modelación de Aplicaciones Educativas ApEM-H.

Que el grupo desarrollo que implementará la herramienta ApEM-H, utilice a el Eclipse como Entorno de Desarrollo Integrado, para la construcción de las misma.

Utilizar el modelo arquitectónico definido en esta propuesta referente a las herramientas que componen a ApEM-H para la incorporación de una nueva herramienta al ambiente CASE Integrado ApEM-H.

BIBLIOGRAFÍA

"Case Integrados".2007.
http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/tecuapetla_l_fj/capitulo3.pdf

"Integración de herramientas CASE usando Internet, Corba y Repositorios de Metainformación". 2002.
http://www.fing.edu.uy/~ruggia/T5s/iCASE01/CITA02_EDelfino_RepCWM.pdf

"Una Herramienta CASE para ADOO: Visual Paradigm". 2007. http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/LabTr1_VP.pdf

"Plataforma CASE Basada en Componentes para la Docencia de Ingeniería del Software". 2007. http://lsm.dei.uc.pt/ribie/docfiles/txt2003731171752paper-140.pdf

"Métodos de diseño hipermedia".2007.
http://www.dei.inf.uc3m.es/docencia/p_s_ciclo/dsh/practicas/metodos.pdf

"Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional". 2006.

http://dialnet.unirioja.es/servlet/articulo?codigo=2065795

"Seminario de Rational Rose 2001". 2001.
http://kybele.escet.urjc.es/Documentos/ISI/Seminario%20Rose.pdf

"Herramientas para el Desarrollo de Sistemas de Información". 2007.
http://www.inei.gob.pe/web/metodologias/attach/lib615/

"Rational Rose". 2008. http://www.indudata.com/1rational_rose.htm#1

"OOCASE: Object Oriented CASE". 2007. http://www-gris.det.uvigo.es/~avilas/UML/node11.html

"Arquitectura de software". 2007.
http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/arquitectura_soft.asp

"Architect Academy: Seminario de Arquitectura de Software". 2007.

REFERENCIA BIBLIOGRÁFICA

[1] “ApEM – L como una nueva solución a la modelación de aplicaciones educativas multimedia en la UCI”. Ing. Febe Angel Ciudad Ricardo. 2007

[2]”Colección Cultura Informática”. Instituto Nacional de Estadística e Informática de Perú. 1999

[3] “Arquitectura y proceso para la construcción de ambientes CASE integrados”. 2007. http://www.lisi.usb.ve/publicaciones/05%20herramientas/herramientas_12.pdf

[4] “Herramientas CASE”. Facultad de Ingeniería. Escuela de computación, análisis y diseño de sistemas. http://rcruz0423.galeon.com/docs/case.pdf

[5] “Análisis y Diseño asistido por ordenador: CASE”. 2004 http://kybele.escet.urjc.es/documentos/HC/HC4GL2007-T1-CASE-Compl.pdf

[6]”Ingeniería de software y su relación con las herramientas CASE” http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rea_c_ji/capitulo2.pdf

GLOSARIO DE TÉRMINOS

CASE: Ayuda por Computadora a la Ingeniería de Software.

TECNOLOGIA CASE: Una tecnología del software que mantiene una disciplina de la ingeniería automatizada para el desarrollo de software, mantenimiento y dirección de proyecto, incluye metodologías estructuradas automatizadas y herramientas automatizadas.

HERRAMIENTA CASE: Una herramienta del software que automatiza (por lo menos en parte) una parte del ciclo de desarrollo de software.

SISTEMA CASE: Un conjunto de herramientas CASE integradas que comparten una interface del usuario común y corren en un ambiente computacional común.

KIT de HERRAMIENTAS CASE: Un conjunto de herramientas CASE integradas que se han diseñado para trabajar juntas y automatizar (o proveer ayuda automatizada al ciclo de desarrollo de software, incluyendo el análisis, diseño, codificación y pruebas.

METODOLOGÍA CASE: Un automatizable metodología estructurada que define una disciplina e ingeniería como un acercamiento a todos o algunos aspectos del desarrollo y mantenimiento de software.

PUESTO DE TRABAJO para CASE: Una estación de trabajo técnica, diseñada a 32 bits o computadora personal equipada con Herramientas CASE que automatiza varias funciones del ciclo.

PLATAFORMA de HARDWARE para CASE: Una arquitectura de hardware con uno, dos o tres sistemas puestos en línea, que proveen una plataforma operativa para las Herramientas CASE.

ANEXOS

Anexo 1

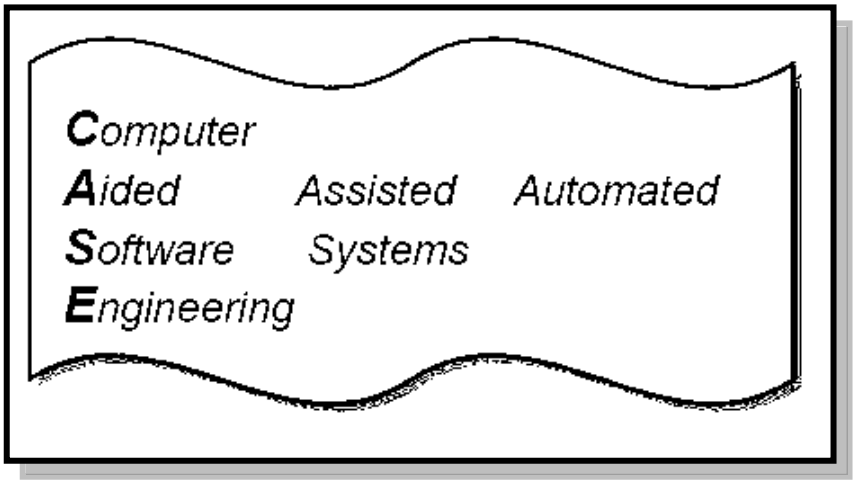


Imagen 1. Variaciones en el significado de CASE.

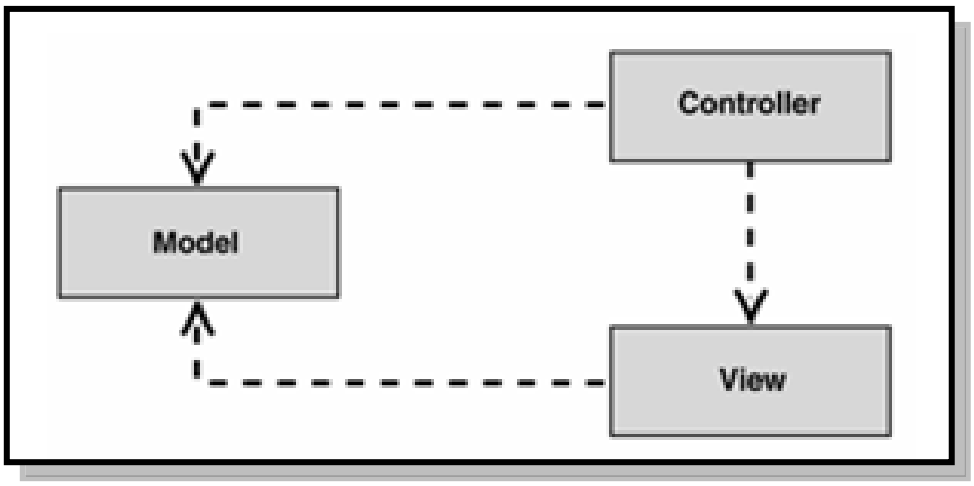


Imagen 2. Modelo – Vista – Controlador.

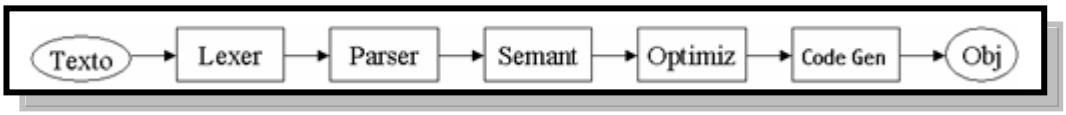


Imagen 3. Compilador en tubería-filtro.

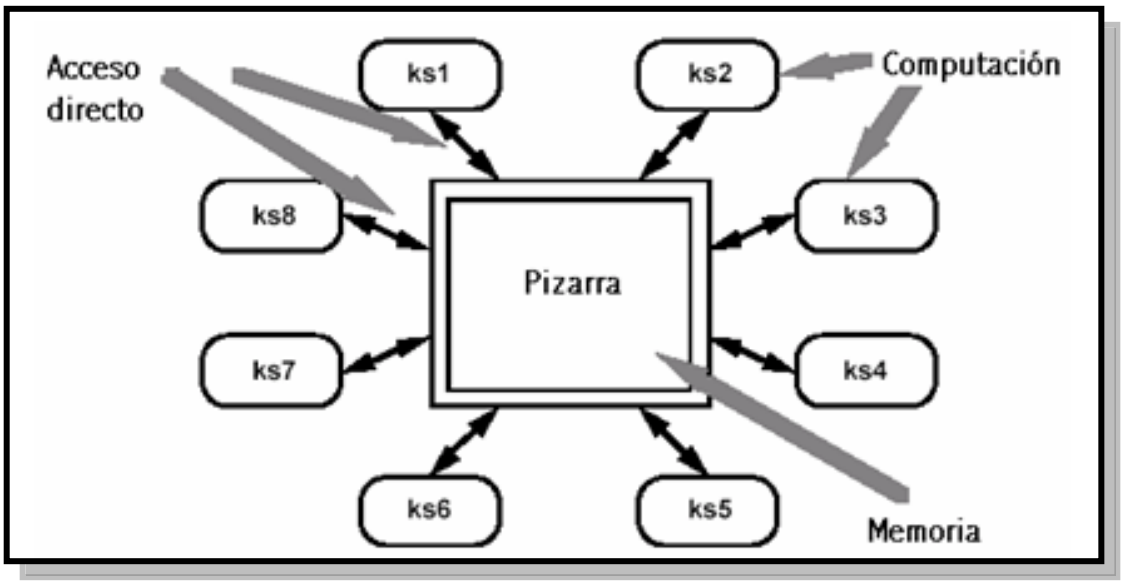


Imagen 4. Arquitectura de Pizarra.

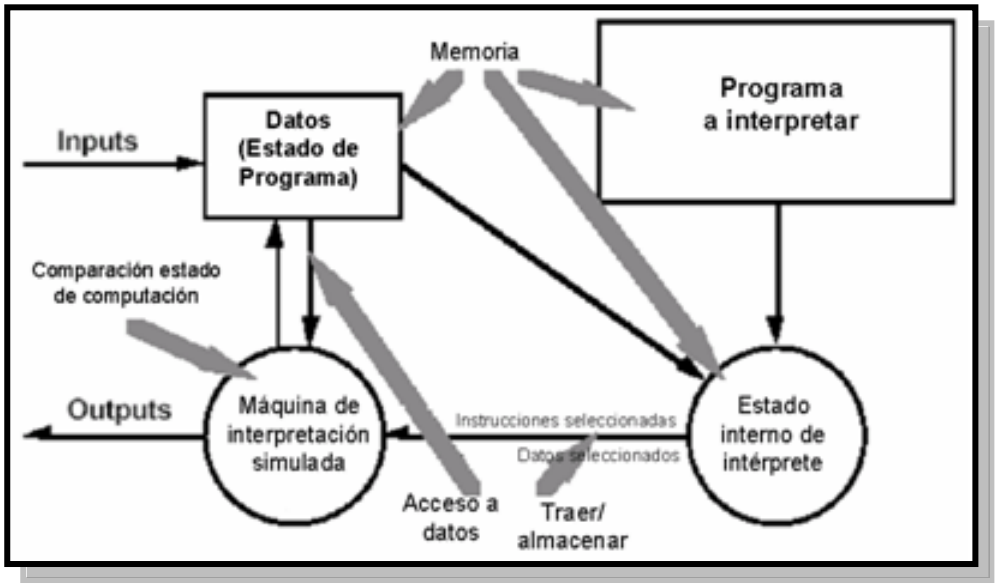


Imagen 5. Arquitectura de Máquinas Virtuales.

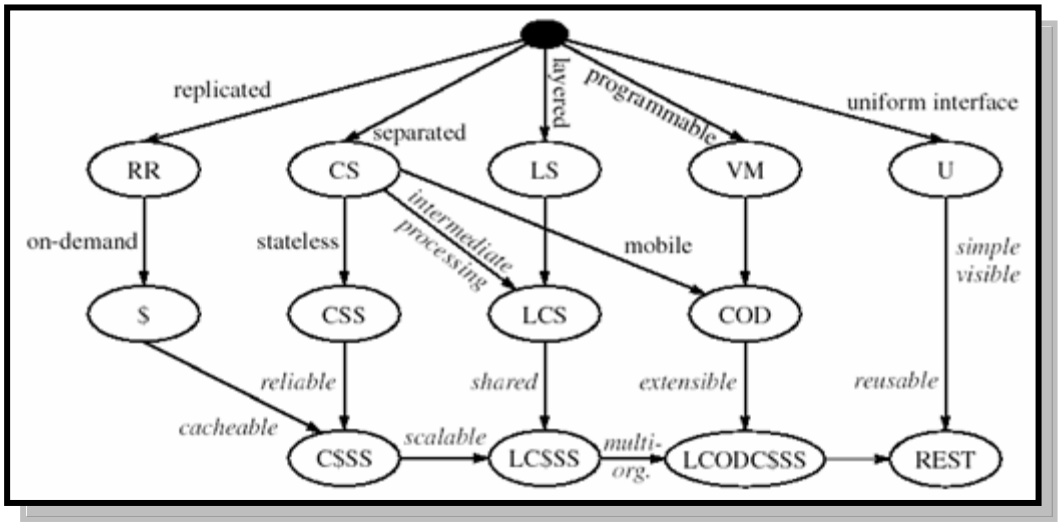


Imagen 6. Composición de REST.

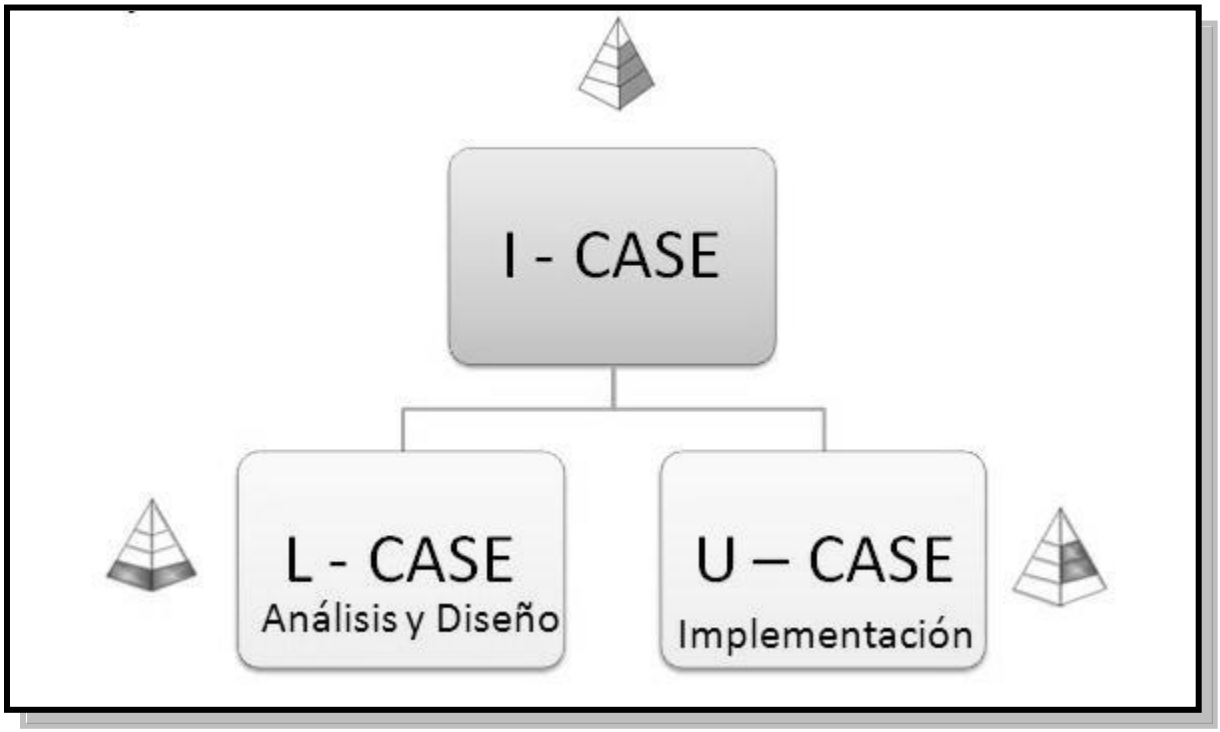


Imagen 7. Fases que cubre ApEM-H.

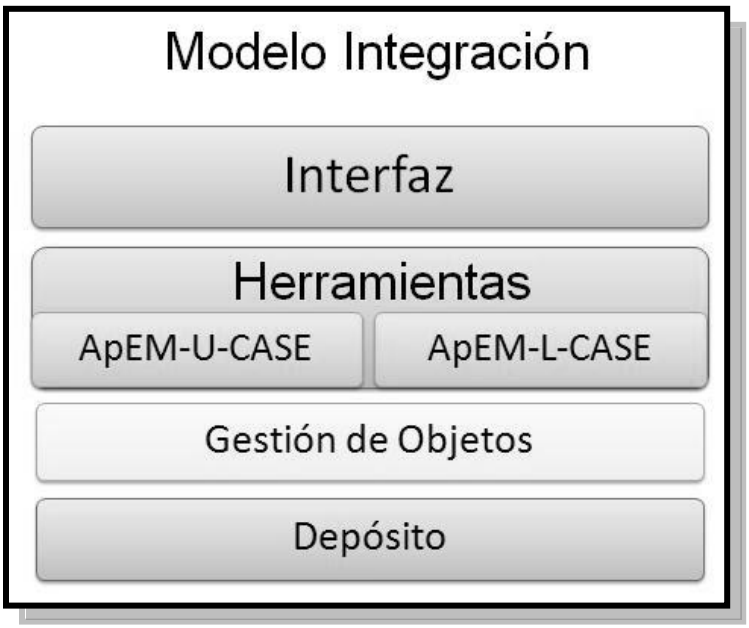


Imagen 8. Modelo de integración de cuatro capas.

Anexo 2.

Tabla 1. Comparación entre el desarrollo tradicional de software, desarrollo utilizando una herramienta CASE y una herramienta I-CASE.

Desarrollo Tradicional	Case	I – Case
Énfasis en la codificación y pruebas de programas.	Énfasis en análisis y diseño.	Énfasis en el modelamiento empresarial.
Especificaciones basadas en papel.	Especificaciones basadas en diagramas automatizados.	Especificaciones basadas en diagramas automatizados y perfectamente integrados a través de las diferentes etapas de desarrollo.
Codificación manual de programas.	Generación automática de códigos, programas fuente.	Generación automática de códigos y totalmente integrada con la estación de trabajo de diseño representación gráfica de las estructuras de códigos.
Documentación manual.	Generación automática de documentación	Generación automática de documentación.
Mantenimiento de programas fuentes.	Mantenimiento de especificaciones de diseño y regeneración de códigos.	Mantenimiento de especificaciones de diseño y regeneración de códigos.

Tabla 2. Comparación de las herramientas CASE de acuerdo a la fases del ciclo de vida que soportan.

Tipo de Case	Ventajas	Desventajas
I – Case	• Integra el ciclo de vida. • Permite lograr importantes mejoras de productividad a mediano plazo. • Permite un eficiente soporte al mantenimiento de sistemas. • Mantiene la consistencia de los sistemas a nivel corporativo.	• No es tan eficiente para soluciones simples, sino para soluciones complejas. • Depende del Hardware y del Software. • Es costoso.
Upper Case	• Se utiliza en plataforma PC, es aplicable a diferentes entornos. • Menor costo	• Permite mejorar la calidad de los sistemas, pero no mejora la productividad. • No permite la integración del ciclo de vida.
Lower Case	• Permite lograr importantes mejoras de productividad a corto plazo. • Permite un eficiente soporte al mantenimiento de sistemas.	• No garantiza la consistencia de los resultados a nivel corporativo. • No garantiza la eficiencia del Análisis y Diseño. • No permite la integración del ciclo de vida.

Tabla 3. Resultado del trabajo de expertos.

G	C/E	E1	E2	E3	E4	E5	E6	E7	Ep
50	C1	12	14	13	12	14	12	13	12.85
	C2	13	12	13	12	13	13	12	12.57
	C3	13	12	12	13	12	12	13	12.42
	C4	12	12	12	13	11	13	12	12.14
25	C5	12	13	13	14	12	13	14	13
	C6	13	12	12	11	13	12	11	12
25	C7	13	13	14	12	13	14	13	13.14
	C8	12	12	11	13	12	11	12	11.85
T		100	100	100	100	100	100	100	100

Tabla 4. Cálculo de concordancia de Kendall.

Expertos/Criterios	**E_1**	**E_2**	**E_3**	**E_4**	**E_5**	**E_6**	**E_7**	**∑E**	**Ep**	**ΔC**	**ΔC^2**
C_1	12	14	13	12	14	12	13	90	12.85	2.5	6.25
C_2	13	12	13	12	13	13	12	88	12.57	0.5	0.25
C_3	13	12	12	13	12	12	13	87	12.42	0.5	0.25
C_4	12	12	12	13	11	13	12	85	12.14	2.5	6.25
C_5	12	13	13	14	12	13	14	91	13	3.5	12.25
C_6	13	12	12	11	13	12	11	84	12	3.5	12.25
C_7	13	13	14	12	13	14	13	92	13.14	4.5	20.25
C_8	12	12	11	13	12	11	12	83	11.85	4.5	20.25
DC	100	100	100	100	100	100	100	700	100	22	78
M ∑E	87.5										
W	0.038										
X^2	1.862										

Tabla 5. Calificación de cada criterio.

Criterios	**Calificación (c)**					**P**	**P × c**
	1	**2**	**3**	**4**	**5**		
C_1				X		0.1285	0.512
C_2				X		0.1257	0.525
C_3					X	0.1242	0.62
X						0.1214	0.48
C_5					X	0.13	0.65
C_6					X	0.12	0.6
C_7				X		0.1314	0.52
C_8					X	0.1185	0.59

Anexo 3.

Modelo No. 1. Guía para informar el peso de los criterios.

Guía para informar el peso de los criterios.

Fecha de recepción _____

Fecha de entrega _____

Nombre y Apellidos del evaluador _____

Le otorgará un peso a cada criterio de acuerdo a su opinión y el peso total de cada grupo debe sumar:

Grupo No.1……50

Grupo No.2……...25

Grupo No.3…….25

Para que el peso total asignado sea 100.

Grupo No.1:

- Representación de las funcionalidades que debe realizar la capa de interfaz.
- Representación de las funcionalidades que debe realizar la capa de herramientas.
- Representación de las funcionalidades que debe realizar la capa de gestión de objetos.
- Representación de las funcionalidades que debe realizar la capa de depósito de datos.

Grupo No 2:

- Definición de la arquitectura de la herramienta ApEM-U-CASE.
- Definición arquitectura de la herramienta ApEM-L-CASE.

Grupo No 3:

- Representación de las características de un OOCASE en especial su arquitectura.
- Aplicabilidad de la arquitectura para la construcción de ApEM-H.

Modelo No. 2 Guía para la evaluación.

Fecha de recepción _____

Fecha de entrega _____

Nombre y Apellidos del
evaluador_____

- Criterios de medida que se evalúan en una escala de 1 - 5

Grupo No.1:

- Representación de las funcionalidades que debe realizar la capa de interfaz.
- Representación de las funcionalidades que debe realizar la capa de herramientas.
- Representación de las funcionalidades que debe realizar la capa de gestión de objetos.
- Representación de las funcionalidades que debe realizar la capa de depósito de datos.

Grupo No 2:

- Definición de la arquitectura de la herramienta ApEM-U-CASE.
- Definición arquitectura de la herramienta ApEM-L-CASE.

Grupo No 3:

- Representación de las características de un OOCASE en especial su arquitectura.
- Aplicabilidad de la arquitectura para la construcción de ApEM-H.

- Categoría final del proyecto

___ Excelente: Alta novedad científica, con aplicabilidad y resultados relevantes.

___ Bueno: Novedad científica, resultados destacados.

___ Aceptable: Suficientemente bueno con reservas.

___ Cuestionable: No tiene relevancia científica y los resultados son malos.

___ Malo: No aplicable.